INTEREX presents the HP Computer Users Conference

NEW ORLEANS 1992

August 23-27, 1992

Proceedings

Volume 2

Sponsored by **INTEREX**, the International Association of Hewlett-Packard Computer Users

INTEREX

The International Association of
Hewlett-Packard Computer Users

Proceedings

Volume 2

of the

1992 INTEREX
HP Computer Users Conference

in

New Orleans, Louisiana
August 23-27, 1992

Online Backup: the Method Behind the Magic

by Joerg Groessler and David Merit
ORBiT Software
319 Diablo Road, Suite 218
Danville, California 94526
(510)837-4143

Without question, the leading cause of scheduled downtime on HP 3000 computers (or most any computer, for that matter) is system backup. The regular copying of data to magnetic tape or other nonvolatile media is a necessary evil that must be carried out to guarantee recoverability in the event of system catastrophe. Depending upon the amount of data being stored, the speed of the computer and backup device, system load, and other factors, a full HP 3000 system backup can take anywhere from a few minutes to several hours.

Reducing backup time

For many sites, the amount of downtime is prohibitive and therefore measures are taken to speed up system backup. The most common method for doing so is to simply reduce the number of files being backed up on a frequent basis. This is easily accomplished by backing up the full system, say, once a week and each day backing up only files that have changed since the last full backup or daily backup.

Another common method for shortening the duration of system backup is the use of data compression, which can typically reduce the amount of data being stored by fifty percent or more. Other methods, such as storing to disk rather than tape (since disk is faster to access) and storing to multiple backup devices can further reduce overall backup time. Data compression and other methods for reducing backup time are available in some backup software packages, although not in the standard MPE :STORE program.

But for many sites, these ways of improving backup performance are still insufficient for reducing the amount of system downtime to an acceptable level. For some sites with critical uptime requirements a desired level of downtime may be zero, and therefore a backup that can be performed with zero downtime–or something approaching zero downtime–is called for.

Online backup capability

An online backup is conceptually a backup that can be performed while users are still accessing files normally, unlike a conventional backup in which users must exit files while the backup is in progress.

An online backup solution must, at a minimum, allow users unrestricted read and write access to files during the backup. A true online backup solution permits users to do anything they would do while a backup is not running. Also, there should be no requirement for users to do their work any differently to accommodate the backup. Ideally, an online backup should be accomplished with no user interruption.

Before investigating how online backups are implemented, let's first examine why downtime is required for backups anyway. After all, it would seem that because a backup only reads files and does not write to them, it should be able to function while users still have files open in a non-exclusive access mode.

Ensuring backup integrity

Conceptually, it is possible to perform a backup while users are reading and writing files non-exclusively. The problem is that data integrity and consistency cannot be guaranteed for files that are being written to. If users are writing to a file while it is being backed up, writes that occur in a portion of the file that has already been written to tape will not appear in the backup, although writes that occur in a later part of the file will be stored. If the file were a detail dataset, it would likely contain broken chains and other internal inconsistencies upon restore.

Besides the need to assure the consistency of each file in and of itself, uniformity must be enforced for all files being stored. Even if integrity could be guaranteed for the detail dataset in our previous example, the dataset must be consistent with the other datasets in the database. And uniformity must be enforced between the database and all other files in the backup.

Therefore, a fundamental requirement of system backup is the ability to return all files on the system to a common status as of a particular point in time. When performing a conventional backup, this point in time is the start of the backup: on restore, files are returned to their status as of the beginning of the store.

Because there is no guarantee that users will not attempt to access files during a backup, and to permit access to files that are not being stored, each file has an associated bit called the store bit which is locked at the beginning of the backup. Once a file's store bit is locked, user access to that file is disallowed until the store bit is unlocked. Store bits are generally unlocked once the file has been written to the backup media, either on a volume change or when the backup is complete. Store bits of all files being backed up are locked at the beginning of the store, assuring that all files are consistent with one another as of the beginning of the backup.

It is the locking of files' store bits that disallows user access during a backup and which causes the downtime required for system backup. However, failing to lock the store bits of files being stored jeopardizes the integrity of the backup because files may be modified during the backup and will therefore not necessarily be consistent with one another. Only in the event that users are reading and not writing to files that are being stored, or through the use of an online backup solution, may a backup be safely performed without locking store bits. Otherwise, if a single user were to perform a single write to a single file during a store, the integrity of the backup would be compromised. To ensure backup integrity, MPE :STORE unconditionally locks the store bits of all files being stored.

How an online backup works

A software online backup solution basically works by not locking store bits during the backup and using a logging process to capture writes that occur during the backup so they may be posted on restore. File uniformity is enforced by artificially simulating a point in time at which the files are consistent with one another. Users are able to work normally with unrestricted read and write access to files during the backup.

Online backup logging is done by trapping writes that occur on the system and then passing them through so that they may be applied to files. The method by which trapping is performed and the level in the system at which trapping is done has a substantial impact on the performance and resource consumption of the online backup. Of course, trapping must be performed in such a way as not to impact the system adversely and should be unnoticeable. For example, a trapping method that would require program or system modifications or that users run through a particular SL or XL would be unacceptable.

On MPE/V, write transactions are sized the same as the block size of the file. Writes to IMAGE datasets are typically sized at 4 to 8 sectors (1024 to 2048 bytes), with the average system write transaction sized at 3 sectors (728 bytes). On MPE/iX (formerly called MPE/XL), writes are done in pages sized at 16 sectors (4096 bytes). Although writes vary in size and may affect several logical records in a file, we will use the term record throughout this article to describe the unit of a file that is modified by a write operation.

Logged writes are saved in one or more log files (collectively referred to in this article as a log file for simplicity) that are stored on the backup along with the data files, providing a history of write activity to files being stored. On restore, the data files are restored and then the transactions in the log file are applied to the restored files, updating them to reflect their status at a particular point in time. The point in time to which files are returned depends upon the type of logging and recovery performed.

Logging recovery

As with other logging facilities (such as IMAGE transaction logging) online backup logging recovery may be implemented using either a rollback or rollforward technique.

When using a rollback recovery technique to restore from an online backup, all files are returned to their status as of the beginning of the backup. Rollback recovery logically backs out writes that occurred to files between the time the backup started and the time the files were written to tape. A rollback recovery thereby results in a restore status equivalent to that of a non-online backup: the checkpoint at which all files are consistent occurs when the backup is started.

Using rollforward recovery on restore, on the other hand, updates all files to their status as of the completion of the backup. It works by posting writes that occurred between the time the file was stored and the backup finished. A rollforward recovery thereby enforces a checkpoint at the completion rather than at the start of the backup.

Rollback vs rollforward recovery

There are certain advantages and disadvantages to rollforward and rollback recovery with respect to online backups. (It should be noted that online backup software products that perform rollforward and rollback recovery are available on MPE/iX. On MPE/V, only a rollforward recovery online backup solution is available.)

In order to facilitate rollback recovery, before-image logging of data is performed. This is done by trapping writes to files and, before committing these writes, reading the records being overwritten and saving them in a log file. This results in the existing files on the system being dynamically updated during the backup while preserving before-images of each record that has been overwritten in a log file on disk. If multiple writes are done to the same record, that record need only be logged once, since the before-image represents the record at the start of the backup and therefore any subsequent writes to that record are insignificant.

On restore, the data file is restored and any before-image records contained in the log file are posted, rolling the file back to the way it appeared at the beginning of the backup. So in practice, an online backup using before-image logging which begins at 6:00 pm and completes at 9:00 pm will, on restore, return all files to their status at 6:00 pm. As stated, the overall result is equivalent to a non-online backup.

In order to facilitate rollforward recovery, after-image logging of data is done. New writes to a file are echoed in a log file so both the data file and the log file contain the latest image of records written to the file. If multiple writes are done to the same record, the latest write must be saved in the log file, since the after-image logging must always reflect the latest version of all records in the files.

On restore, the data file is restored and any after-image records contained in the log file are posted to the file, rolling its status forward to the end of the backup. An equivalent online backup as in the previous example would, on restore, update all files to the way they appeared at 9:00 pm rather than 6:00 pm.

The main differences between using a before-image logging technique with rollback recovery and after-image logging with rollforward recovery are in functionality, data protection, convenience, resource consumption, and performance. Functionality issues include how much user downtime is required and where that downtime is imposed. Resource issues include disk space utilization and CPU overhead.

Functionality

As stated, an online backup should be virtually unnoticeable and should allow users to go about their normal work without restriction. This means that in addition to handling writes to files, other system dynamics must be accommodated as well. This includes the building of new files, the renaming and purging of existing files, the saving of temporary files into the system domain, and the changing of file attributes.

For a before-image logging online backup, these dynamics are not too much of an issue since only activity involving files that existed at the beginning of the backup needs to be handled.

For an after-image logging backup, however, the process is much more demanding. Files that existed at the beginning of the backup but which were purged during the backup must be disallowed from restore even though they may exist on tape, since they did not exist at the checkpoint imposed at the end of the backup. Files that were renamed during the backup may have been stored under one name but need to be restored under a different name. Files that were created during the backup must be detected and created on restore, even though they did not exist in any form at the beginning of the backup. Temporary files which are saved during the backup must be stored, since they have become permanent and are therefore storable.

These matters are relatively straightforward for a full backup but are more challenging for a partial backup, or any backup that restricts the store to a subset of the files on the system. There is the potential with an online backup which performs after-image logging that some files that did not qualify for store when the backup started will qualify during the backup; likewise, files that qualified at the start of the backup may be disqualified during store.

For example, let's say a partial backup that selects files that were modified on the current date is being performed. After the backup starts, a file that has not yet been modified today and which therefore did not qualify for store when the backup started is opened for modification during the backup. Thereby, the file's modification date is changed to today's date, and the file dynamically qualifies for store.

Another situation is a backup that selects files by name, in which some files may be renamed into and others out of the store fileset, including renaming a file from a group that is not being stored into a group that is. A more extreme example is a backup that selects files by filecode or size in which one or both of these attributes are changed during the backup.

Data protection

Like a conventional backup, a before-image logging backup with rollback recovery on restore returns the system to its status at the beginning of the backup. On the surface, this seems adequate and sensible; however, such a backup introduces a substantial data protection and integrity concern that does not exist with a non-online backup. Namely, users are working during the backup but their activity is not safeguarded until the following night's backup.

In comparison, an after-image logging backup provides data protection for write transactions that occur during the backup, which is a very important recovery issue if users continue to process transactions during the backup. Also, because most sites begin nightly batch processing following the backup, it is critical to have a consistent backup reflecting the status of the system immediately before beginning night processing. This guarantees that the system can be returned to this state should there be a problem with night processing; otherwise, the system will be rolled back to a state in which user transactions are lost and must be somehow recreated before night processing can be restarted.

Convenience

In order to facilitate and enforce the consistency of an online backup, exclusive access to files for a brief period of time is generally required. This point in time forms a checkpoint and is referred to as the synchronization point. The amount of downtime is dependent on the number of files being stored and can range from about two to forty minutes, with an average time of about ten minutes.

When using a before-image logging technique, the synchronization point is imposed at the beginning of the backup; for after-image logging, the synchronization point is at the end. So users need to log off either when the backup starts or when the backup completes. The difference is a matter of preference.

A synchronization point at the beginning of the backup may be easier to schedule because it occurs at an absolutely known time, and it may be desirable to have a regular time in which users always need to log off for backup. Sites that have users working on established shifts may find it possible to begin the online backup during a shift change, thereby minimizing its impact.

Having the synchronization point at the end of the backup is typically less of an impact because users can continue working when the backup starts. Generally, fewer users tend to be working upon completion of the backup and therefore fewer users need to be interrupted overall. Also, the task of having users exit their files when the synchronization point occurs at the end of the backup may be delayed to allow users to continue working until a convenient, or a specific, time: logging continues until the backup utility is notified that it is ok to proceed with synchronization.

Resource consumption

The disk space requirements for a before-image logging online backup vary based on the locations of the writes being performed to files. For example, if an entire file is written to, such as an IMAGE dataset in which a program is performing DBUPDATEs throughout the dataset, an entire before-image copy of the dataset may need to be built and maintained on disk during the backup. On the other hand, if only some entries are appended to the dataset, the disk space requirements will be greatly reduced. In the worst case, the disk space requirements for logging can be equal to the amount of disk space consumed by the files being stored. This, of course, assumes that every record (or whatever defines the incremental size of a write transaction) is written to. However, with more activity during an online backup–which is a function of the number of users logged on, the duration of the backup, and the location of the writes–quite substantial disk space may be needed.

An after-image logging backup, on the other hand, can require far fewer system resources–especially in the area of disk space utilization. In fact, it can require almost no disk space and impose very little CPU overhead. This is because a very different technique can be used for logging. Since the requirement for a rollforward backup method is that only the latest version of each modified record be logged for recovery, that record is always available within the disk files themselves at the completion of the backup, and it is therefore sufficient to simply keep track of the writes that occur within files during the backup and then fetch the modified records upon completion of the backup.

So rather than capturing and maintaining a before-image of modified records on disk during the backup, it is sufficient to keep track of which records have been modified during the backup and then, once all files have been written to tape, to read those records from the files on disk and write them directly to tape. In effect, the data files themselves act as log files so there is no need for a separate logging area on disk during the backup.

In summary, a rollback recovery backup needs to keep the actual before-image record of each modified file on disk before it is later written to tape, and this can in the worst case require an overhead of as much disk space as the files being stored currently occupy. A rollforward recovery backup, however, requires virtually no disk space because it does not need to log the after-images to disk at all.

Besides the savings in disk space, another advantage of using an after-image logging technique is that the high CPU overhead and memory utilization required by before-image logging for data transfer and disk writes is eliminated.

Performance

Issues of performance and resource consumption go hand-in-hand since the fewer resources consumed, the better the overall performance.
The primary issue affecting performance–for both rollforward and rollback recovery methods–is the volume of writes that occur during the backup and where they occur. In any case, write activity during the backup should be minimized since even if there is little overhead in system responsiveness, there is more data being written to tape and therefore the backup takes more time and media.

The performance penalty for an online backup versus a non-online backup can be as high as fifty percent for a rollback recovery tool while performance degradation for an rollforward recovery tool can approach ten percent, according to the vendors of online backup packages.

Conclusion

Whereas most computer system platforms do not offer a software online backup solution, software online backups are commonplace on the HP 3000. Online backups are in daily use at hundreds of sites throughout the world with excellent results.

Experience shows, perhaps unexpectedly, that online backup capability is of interest not only to sites with obvious high availability requirements (hospitals, police departments, etc.) but also to companies whose users sometimes work late or that provide system access to users in different time zones. Many sites use online backup capability simply to start the backup in the afternoon so the operator can go home with the rest of the users while knowing the system data is safely stored on tape.

Online backup is a sophisticated software solution to a universal problem and takes the greatest pain out of system backups–not only for Operations but, more importantly, for the users it serves.

Joerg Groessler is the creator of Online-BACKUP/3000 and President of ORBiT Software, Danville, California; David Merit is ORBiT's Vice President.

Paper 3058
A Plan for Input and Output Queues

Terry Warns
WL SOFTWARE
1281 Bradbury
Troy, MI 48098
1-800-397-7097 (US Only)
1-313-641-0550 (World)

In my travels as a software developer, I notice that most organizations use the default values as provided by the manufacturer. Even bigger shops with 950's and 980's have never developed a strategy to utilize features of the MPE operating system. Many times companies will even buy a package for thousands of dollars when standard MPE could provide a good solution. In this light I am going to discuss some simple yet effective solutions using MPE and controlling your input (Jobs) or output (spool) queues.

In these days of GUI, WINDOWS, X-WINDOWS, MOUSE, PC's, and other buzz words, we still need batch and we still need spool files. It is not as glamourous as pull down menus or pop up selection lists or buttons. In fact, I doubt we will ever get rid of batch processing. We still need to backup. We still need to produce a general ledger trial balance that is hundreds of pages long. MRP still takes 4 hours to run. Even in the new concepts of open systems and portability of files and programs, and relational databases, we still need batch.

INPUT QUEUE

The input queue is the queue of batch jobs to run. Once the job starts executing, the input queue has no effect on the job. The input queue only manages jobs waiting to run. I have worked on PC's, Prime, and IBM mainframes. PC's has no equivalent of an input queue. A Prime minicomputer has batch jobs but no input queue. This means if 100 people submitted batch jobs on the system, then 100 batch jobs start to run. IBM mainframes has multiple input queues. The HP has one input queue. I consider the availability of a input queue as a major feature of MPE.

The features of the input queue on MPE are the following:

1. Limit the number of jobs running
2. Limit the priority of jobs to be run
3. Alter the priority of the jobs waiting to run

We limit the number of jobs to run with the use of the LIMIT command of the operator. At the console, you could type:

```
:LIMIT 2
```

You are limiting the number of jobs running to 2. If three jobs are running when you issue the command then the system allows three jobs to run. The next job to start will not begin executing until two of the three jobs are finished. Under normal circumstances, this really means to allow the next job in the input queue to begin executing when the currently running jobs is less than 2.

Why would we do this? Depending on your machine model and your application mix, and your on line performance goals, and the time of day you would not want to run all the jobs at one time. If you had 100 jobs running at once, your computer would get nothing done. It would be swapping out memory all the time. The machine would be at a standstill. So we limit the number of jobs running to promote more throughput.

The second feature to consider is limiting the priority of jobs to run. At the console you could type:

```
:JOBFENCE 8
```

Your are limiting the input queue to allow running only those jobs with a priority of 8 or more. There are 15 priorities: 0 through 13 and HIPRI. If you stream a job with a priority of HIPRI, then that job will run regardless of the job limit or the jobfence. If your job priority is 0 then the job will never run without the operator changing the priority. If your job is less than the jobfence then your job will not run until the jobfence has been changed or the individual job's priority has been changed. These two features of job limit and job fence becomes a very powerful tool in our strategy.

In order to submit a job you must create a MPE file and use the STREAM command. The command is the following:

```
:STREAM filename
```

"filename" can be a temporary file or permanent file. A job number is provided to you at the time of streaming. The first line of the filename is a JOB card. Each subsequent line is a standard MPE command or UDC or command file name. Instead of the a system supplied colon, you must provide a ! instead. An example of a job follows:

```
!job manager.sys
!listf
!eoj
```

This is a job that logs on as MANAGER.SYS, does a LISTF and exits the system. "!job" is equivalent to HELLO for a session. If the logon user,group or account has password you must provide them on the job card. The default priority is 8 for jobs. This job would go into the input queue as the last priority 8 job and wait its turn based on the job limit and jobfence. This job would be run before any priority 7 jobs no matter when they were submitted. It is a simple FIFO priority scheduling system. If I wanted to change the priority to 9, I would modify the job card as follows:

```
!job manager.sys;inpri=9
```

With this simple addition to the job card, I changed its place in the job queue. At the time of creating the job I can predetermine the job priority I wish to put this job into. I just need to establish a system to the job priorities that can help me schedule my jobs.

STRATEGY:

The default priority is 8. We will assume that most jobs will be of default nature. In this way we can begin to build our strategy and start to describe the "default" job. If the default is 8 we must have jobs that have lower priority than the default. For instance let's say programmers compiles and tests are lower in priority than production reports. We can assign programmers jobs to a priority of 7 and production reports will use the default. This simply assures that programmers jobs will only start running if all the production reports are finished or running. We can set the jobfence to 7 rather than the default 8 and the above statement will be true. Now we can change any job to be less priority than "normal" jobs.

Similarly, we can identify jobs that have higher priority than normal jobs. For instance, a job that is requested by the CEO is required to be run as soon as possible could have a priority of 9. That job will automatically go in front of the jobs with a priority of 8. There are some jobs that need to be run next no matter the circumstances. Their priority could be 11. I know that there will be times that I wish to override everything so I reserve priority 13 for the operator.

Finally some on line functions stream jobs and wait for jobs to complete. I had a warehouse that needed its picking list immediately so I had that job at HIPRI. It ran no matter the jobfence or job limit. There are jobs that must be run overnight. I use an inpri of less than the daytime's jobfence say 1 or 2. They cannot run during the day but they can sit in the queue waiting all day. Imagine there are two job queues. The day job queue has priority of 7 through 13 and HIPRI. The night job queue has priorities of 1 through 6. Job queue of 0 is deferred until the operator changes it.

DAYTIME JOB QUEUE:

The following is the strategy for the daytime job queue:

Queue	Purpose
HIPRI	Begin Executing immediately
13	Reserved for operator
12	Jobs that support online function
11	High priority jobs
10	Short jobs under 5 minutes
9	Medium priority under 30 minutes
8	Default - jobs under 30 minutes
7	low priority day jobs, or longer than 30 minutes

I see people running jobs during the day that will run for more than 8 hours. If your job runs that long why do you run it during the day? It won't finish until you have left for the day. I consider the run time of jobs running between 5pm and 8AM to be 1 minute. If it cannot finish by 5PM then your next minute of work is at 8AM. During the day I establish a guideline of 30 minutes for a "normal" job. I prefer to have many short jobs finish than no jobs finish because we have a long job running during the day. Your guideline may be longer or shorter.

Also I prefer all 5 minute jobs to complete before the longer 30 minute jobs to run. There will be some 30 minute jobs that have a higher priority than "normal" jobs. The CEO job may have a higher priority than even short jobs. Finally I want online functions to get the best possible priorities in the job queue. I also need to reserve one queue for the operator to utilize in order to override everything I just said.

One major problem exists in this scenario. At some point in time, you could have two long running jobs executing, your job limit at 2 and users streaming priority 8 through 12 jobs. If this happens, the high priority jobs have to wait until the long running jobs are completed. This may take hours. To correct this, you can set your limit up to 3 and suspend the jobs with BREAKJOB. When your queues are cleared up, you can resume the broken jobs. If this is a continual problem, then a job scheduling package may be a good choice.

To ensure good online response and good batch throughput we must limit the number of jobs to run at one time. This limit is dependant upon your unique mix and may take some experimenting to determine the optimal limit. In general a 980/400 would probably handle substantially more jobs at one time than a 917LX. At the low end could be 1 or 2. At the high end you could have 8 or more. Also at the high end you will have fewer long running jobs. At the low end you will have more. As you upgrade hardware, the definition of the queues may change and the priority of some jobs may change.

NIGHTTIME JOB QUEUE:

The strategy for night job queues is as follows:

Priority	Purpose
6	Reserved for operator
5	Jobs require operator
4	Jobs running by themselves
3	High priority jobs
2	Medium priority jobs
1	Low priority jobs
0	Permanently Deferred Jobs

Priority 6 is reserved for the operator in order for him to schedule jobs that he/she needs to rearrange or run first. The major checkpoint of the night is the backup. For a backup, you wish only the backup job to run. Therefore your job limit must be 1 for a successful backup. The job priority for a backup is 4. Typically, the night is complete for the operator after the backup. You do not want an operator sit for 2 hours while some job runs and he/she has nothing to do but watch the lights. The jobs prior to the backup may require operator intervention. Tape requests or payroll checks are good examples of operator intervention. We want those to run before the backup. Once the backup is complete we want other jobs that require themselves to run standalone. MRP could be required to run by itself. At this time of the night the job limit would be 1 and the jobfence would be 4. After the last priority 4 job, you can group your jobs into high , medium, and low priority. Since we have eliminated standalone jobs, we can set our limit to more than 1. Our jobfence can be set to 1.

Sometimes jobs cannot start until other jobs are completed. Third party packages provide features that control this situation. In our strategy we will stream jobs from within other jobs. Our priorities and fences still apply.

I suggest that you avoid scheduling jobs at a certain time. First off the schedule time is only when the job is release to the job queue not when it starts running unless you have hipri. It may be useful to delay the start of some jobs until after a certain time at night. Unless you have substantial online activity at night, then I consider the night hours batch job time. I have twelve to fifteen hour time frame to get things done. I do not want to have idle time during the night and not meet my morning deadline. Therefore, I avoid scheduling by time and schedule by priority.

In conclusion, our goals for the execution of batch jobs are different during the day than at night. During the day, we wish to schedule jobs based on run times. More short jobs is better than a few long jobs. At night, we are interested in optimize the batch throughput and only have idle time when there are no more jobs to run. By establishing rules about priorities, we can use standard MPE to help us manage automatically these goals.

One buzzword in the industry is to be operatorless. These standards allow a company to approach being operatorless. First area of concern is backup. If we can backup without operator intervention, we can begin to go operatorless. If we have already identified the jobs which require operator intervention, then we can work on the eliminating that intervention. All other jobs are running operatorless.

OPERATORLESS

The big problem with most shops face is how to do backups without an operator. There are some third party tools that can help you. We however will suggest how you can utilize standard MPE to potentially accomplish this goal. First the new DAT tapes handles around 1G of data. If you are backing up less than 1G then, you can use this tape drive to go operatorless. Many organizations require more than 1G everyday.

Your first inclination is to backup everything everyday. It is the safest, most complete, easiest, and most time consuming method available. If we assume you use IMAGE datasets for your critical data, we could just backup the sets that were modified for the day. However, consider your a 2G dataset that you changed 6 characters with. Do you really want to use 2 DAT tapes and over 30% of your batch time to record a 6 character change? Of course not. Instead, log your Image datasets and backup your log files. Then weekly or monthly do a full IMAGE database backup. Second only backup non database files if they have been changed. Third backup KSAM or other data files with an internal logging mechanism or put the changes into IMAGE database which is then logged. Ideally we wish only to backup our changes rather than backup everything. If we can accomplish this goal, then only the very largest of companies are generating 1G of new or changed data every day. Standard MPE provides you the opportunity to go operatorless.

OUTPUT QUEUE

The output queue is similar to the input queue. We can set priorities to individual files and we set fences to prevent the file from printing. We also have multiple print queues for any printer.

First off realize that PC's do not have spool file capabilities although some PC packages do. Second realize that IBM mainframe capabilities are a lot more significant than HP's capabilities.

HP spooling is a FIFO priority based system. The files are given a priority and the spool files printed in that priority in a FIFO manner. I recommend that you would want all small spool files first and then larger spool files after the small ones. In order to accomplish this, we must establish the priority of the spool file before it is printed. We can provide a higher priority for expected small files and a lower priority for large files. If 8 is our default, then priority 9 may have small reports (less than 10 pages). Priority 7 may have large reports (more than 250 pages). We establish our lower fence to be 7. Simply we have broken our reports into three categories, small, normal, and large.

Using the same logic as the input queues, we can establish HI priority print files at 10,11 or 12. We can reserve 13 as the operator's only queue. Also we can prevent files from printing.

We establish priority 2 for all job stdlists. We establish 6 as super large reports to be scheduled. We establish priority 3 as special forms that require operator intervention. Since we can have multiple printers or sets of printers, we can have different priorities for different printers (Queues).

SPECIAL FORMS:

I have seen companies put on the special forms to a printer when they scheduled the job to run so that all printouts will utilize that form. I believe all special forms are put into 1 priority. When the operator is ready to print the file, he/she changes the priority to 13. If a forms message is used, a message on the console is displayed and a test line is printed. If a forms message is not used, then the outfence is set to 14 so that nothing prints. When the special form is ready , the outfence is set to 13. The special form is printed, and the outfence is set to its normal setting.

If we look at going operatorless, we can print special forms on laserjet and eliminate this operator intervention.

STDLISTS

We do not like to throw out stdlists. We save them with a priority of 2. We can then print all stdlists together or we can save them to tape.

MICROFICHE

If you produce microfiche tapes from spool files, I would establish a priority below the standard list. All microfiche tapes would be created from this priority.

OUTPUT QUEUE STRATEGY:

Output Queue	Purpose
13	Reserved for Operator
12	
11	Hi priority reports
10	Screen Lists 1 page
9	Small Reports < 10 pages
8	Normal Reports < 100 pages
7	Large reports > 100 pages
6	Large reports > 500 pages (scheduled)
5	
4	
3	Special Forms
2	Job Standard lists
1	Microfiche Reports
0	Never printable

We are trying to manage a possibly limited resource, the printers. If your printers are never backed up, then you may not have the problem. Many printers do not begin printing the nightly jobs until the morning. We need some way to priorities the printouts. In this scheme we priorities the reports by size. You could use other criteria. You need to reserve one queue for the operator to override everything. Screen lists are important to be printed quickly. Special forms need to be easily identified by the operator. STDLISTS needs identification for support people and kept around on paper, fiche, or tape. Microfiche reports need to be easily identified. These last three require operator intervention. Finally, we may not want to print certain reports at all.

In conclusion, we are trying to manage a limited resource and set some priorities. We are trying to manage this area within standard MPE.

CONCLUSION:

This paper was written for new systems managers. We avoided third party solutions. We used standard MPE commands and features. We created meaning for the queues. We gave examples of why you may want some jobs to run before others. We gave examples of why you may want some printouts before others. We showed you a daytime strategy and a nighttime strategy for jobs. We explain some background on reducing your backup requirements. We did this without any additional costs on your part.

We realize each shop has different priorities for itself. Therefore the suggestions will rarely be implemented as presented. One large shop has hundreds of jobs every night and does no prioritizing of jobs. Typical problems are tape drives idle for hours, unanswered tape requests, and multiple requests for the only tape drive at the same time. They fail to complete their jobs on time. They say cannot priorities the jobs. This strategy will work if those problems are typical. Third party tools can help in scheduling jobs by the day, by the time, and most importantly by interdependence of jobs. However your problem must not be able to be satisfied by this strategy. If you have no strategy, then these strategies are good start.

PAPER NO.: 3059

TITLE: Client/Server Architecture for MPE/XL-HP-UX Macintosh Windows

AUTHOR: James F. Dowling
Bose Corporation
The Mountain
Framingham, MA 01701-9168
(508) 879-7330

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO.: 3061

TITLE: Glance/XL Advanced Analysis

AUTHOR: Michael Hornsby
Beechglen Development, Inc.
5576 Glenway Ave
Cincinnati OH 45238
(513) 922-0509

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

Adopting Self-Describing Files

By David J. Greer

Robelle Consulting Ltd.
Unit 201, 15399-102A Ave.
Surrey, B.C. Canada V3R 7K1
Phone: (604) 582-1700
Fax: (604) 582-1799

Abstract

Query can generate output to self-describing (SD) files, but no HP products read these files except the old DSG and Listkeeper products. A self-describing file is a data file which stores a standard description of its own record format in its user labels. Thus, it is like a little stand-alone database (a trendy developer might call this object-oriented). If two software tools understand SD files, it becomes trivial to transfer data between them. A user can archive some data in an SD file and when it is restored five years later the SD file can tell what the data means. The author describes the internal format of SD files, gives examples on how to read and write SD files, and describes problems integrating SD files into a software tool.

Introduction

For years, Query's Save Command has been able to create a file that is self-describing. A self-describing file is one that contains the information about the fields in the file. Normal MPE and KSAM files are not self-describing. In general, we know nothing about the structure of the fields in each record.

Unfortunately, few software tools create or understand self-describing files. While Query can produce self-describing files, it cannot use them as input. Our product Suprtool can both create and understand self-describing files (including KSAM ones). In addition, Suprtool has a new self-describing format that removes some restrictions of the original self-describing structure. In this article we will do the following:

- Describe the format of both the original self-describing file (this will be a summary of the information in Appendix E of the Query User Manual) and the new Robelle self-describing file.
- Show how to create a self-describing file.
- Give a programming example that can understand and provide a "form" listing of any self-describing file.
- Describe KSAM self-describing files.
- Speculate on what an "open system" self-describing file would look like.

Query Versus Robelle Self-Describing Files

Throughout this article we will refer to one of two types of self-describing files. The first kind are equivalent to the ones produced by Query. They are identified by the version number " A.00.00". The second kind were designed to overcome limitations with the original self-describing format. We identify the revised files by calling them Robelle self-describing files. They have a version number of " B.00.00".

Examples In This Article

Because we write code in SPL and SPLash!, we will give our examples in these programming languages. The only word of caution is to remember that SPL uses zero-based addressing for all its arrays.

MPE User Labels

User labels are an optional part of an MPE file. User labels are part of the file, but they are not part of the data (i.e., when reading the records in the file the user labels are ignored). User labels are a handy place to store extra information about a file. Unfortunately, MS-Dos and

UNIX have no concept similar to MPE's user labels (see the section *Future Self-Describing Formats* for ideas for UNIX and MS-Dos).

The number of user labels must be specified when the file is created. On most versions of MPE, the only way to create a file with user labels is via the FOPEN intrinsic. Newer versions of MPE/iX allow the ULABEL= keyword on the Build Command to specify the number of user labels. Each user label is 256 bytes long and user labels are numbered from zero. You access user labels by calling the FREADLABEL and FWRITELABEL intrinsics.

Identifying Self-Describing Files

An MPE file that is self-describing has a filecode of 1084. You will recognize these files by seeing "SD" next to the filecode of a :listf,2:

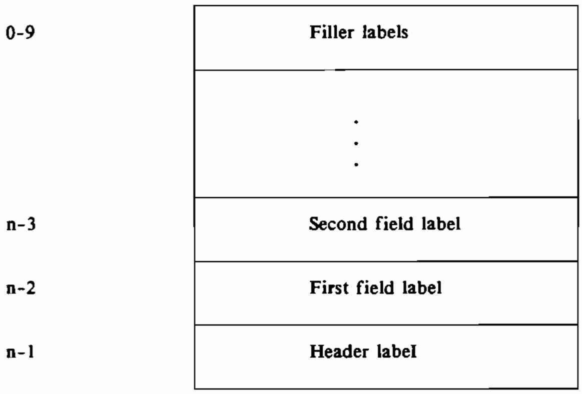
```
FILENAME  CODE  ----------LOGICAL RECORD---------  ----SPACE----
                  SIZE  TYP     EOF      LIMIT R/B  SECTORS X MX

LOADFILE  SD      128W  FB       33      10000  35      256  1  *
```

Recognizing self-describing KSAM files is more difficult. KSAM sd-files do not have a special file code. Instead, you must look for a KSAM file with extra file labels. On MPE/iX, this is done with a :listf ,3 (on MPE V/E use Listdir.Pub.Sys):

What Is A Self-Describing File

A self-describing file stores information in the MPE file labels about the fields in each record of the file. File labels are like a special file within a file. An MPE file label is 256 bytes long and an MPE file is created with 0 to 256 file labels. The file labels are accessed via the Freadlabel and Fwritelabel Intrinsic. User labels are numbered from zero.

Customarily, tools that create self-describing files leave the first ten file labels (numbered 0 to 9) for user applications. The self-describing information is broken into two kinds of labels: the header label and field labels.



Header Label

If an MPE file has n file labels, they are numbered from 0 to n-1. The self-describing labels are always added at the end of the any file labels needed by the user. The last file label will be the sd-header label and the sd-field labels are arranged backwards from this label (n-2, n-3, ...). The format of the header label is similar, but different for Query and Robelle self-describing files.

Query Header Label

The Query header label consists of the following fields:

version (X8). Always equal to " A.00.00" for Query self-describing files.

length (J1). The length of each record in the file in bytes. It appears to always be identical to the MPE record length of the file.

fields (J1). The number of fields in each file record.

labels (J1). Number of labels used for field descriptions plus one for the header label. This is different than the number of MPE labels for the file.

fields'per'label (J1). Each field label contains one or more field descriptions. Do not assume a fixed number for this field -- you must check the value of this field.

size (J1). Length of each field descriptor in 16-bit words. Because MPE file labels are always 128 words long, the **fields'per'label** should always be 128 / **size**. Again, do not assume a fixed constant for the field descriptor size.

Robelle Header Label

The Robelle header label contains all of the fields of Query's header label with one change (the version number is different) and three additions:

1. The version number is " B.00.00" instead of " A.00.00" (note the space at the beginning).

2. There are three new fields for handling sort keys. These fields are identical to the fields that you would pass to Sortinit (in compatibility-mode):

   sort'max'keys (J1). Maximum number of keys allowed in this sd-file. The **sort'keys** would be declared as:

   ```
   integer array sort'keys(0:sort'max'keys*3-1)
   ```

   sort'num'keys (J1). The actual number of keys in the table. This value must range from zero to **sort'max'keys-1**.

   sort'keys. The sort keys themselves using the same conventions as Sort/3000. The byte-offsets of each key start at one and not zero in the sort table. The byte offsets in each field entry remain the same (i.e., zero-based instead of one-based offsets). The sort key types correspond to those for the Sortinit intrinsic and not the newer HPSortinit.

SPL Layout of the Header Label

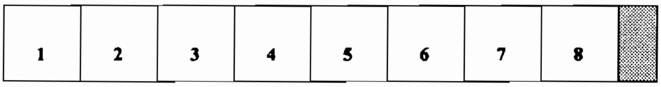
Here is the layout in SPL notation of the Robelle header label. Note that we use exactly the same layout for accessing Query header labels (we just ignore all sd'sort'... variables when accessing Query self-describing files). Each field descriptor is fifteen words long, but even the Robelle field descriptor only uses fourteen words. We leave the last word unspecified (our code always sets the filler words with binary zeroes):

```
sdheader.srcinc:

integer array sd'header(0:sd'label'len);   { 0 : 127 }
byte    array sd'version(*)          = sd'header;
integer array sd'reclength(*)        = sd'header(4);
integer array sd'numfields(*)        = sd'header(5);
integer array sd'numlabels(*)        = sd'header(6);
integer array sd'fieldsperlabel(*)   = sd'header(7);
integer array sd'entrylen(*)         = sd'header(8);
integer array sd'sort'max'keys(*)    = sd'header(9);
integer array sd'sort'num'keys(*)    = sd'header(10);
integer array sd'sort'keys(*)        = sd'header(11);
```

Field Labels

Every self-describing file has one or more field labels of 256 bytes. Each field label has one or more field descriptors. The first fields in the file will be described in label N-2, the next set of fields in N-3, and so on. This is opposite to what you might expect. Self-describing files that Query produces have eight field descriptors per user label.

Picture of one field label



Query Field Labels

Query always produces self-describing files with 15 words reserved for each field descriptor. Each field is described as follows:

field'name (X16). The name of the field left-justified. Field names are in upper case.

field'type (J1). The type of the field taken from the following list:

1. ASCII (type U and X).

2. free form ASCII numbers.

3. signed integer (type I).

4. floating point real (type R).

5. packed decimal (type P).

6. COBOL computational (type J).

7. unsigned integers (type K).

8. zoned decimal (type Z).

9. IEEE floating point (type E). This is a Robelle extension that applies to either " A.00.00" or " B.00.00" self-describing files.

10. IMAGE compound field.

field'offset (J1). The offset of the field in bytes. The offset starts at zero.

field'length (J1). The length of the field in bytes.

reserved'space (4J1). Four words that are reserved for future use.

Robelle Field Labels

Many HP 3000 applications contain repeated fields. Query self-describing files map all repeated fields into type "10", which is useless for applications that understand repeated fields. It would also be nice if additional user information, such as the number of decimal points or the format of a date were available. The Robelle field descriptor provides for all of these, by using three of the four words of reserved space. All fields up to **field'length** are the same as QUERY's (note especially that **field'length** is the total length of the field and not the length of one sub-field). These are the new fields:

field'repeat (J1). In IMAGE terms, this is known as the sub-count. For simple fields, **field'repeat** is one (and not zero).

field'decplaces (J1). Logical number of decimal places in the field. Zero means there are no decimal points. This field must be zero if the **field'type** is byte.

field'date'type (J1). Zero if the field is not a date. Otherwise, contains a constant that describes the format of the date. These constants are described below.

reserved'space (J1). One word that is reserved for future use.

Date Format

The date format is mapped into the data type and byte-length of the field. Here are the constants for each date format:

1	yymmdd
2	ddmmyy
3	mmddyy
4	yymm
5	calendar (MPE intrinsic format)
6	yyyymmdd

7	ddmmyyyy
8	mmddyyyy
9	phdate (PowerHouse format)
10	ask (ASK ManMan format)

SPL Layout of the Field Descriptor

Here is the layout in SPL notation of the Robelle field descriptor. Note that we use exactly the same layout for accessing Query field descriptors (we ignore the repeat, decplaces, and date'type fields for Query self-describing files):

```
sdfield.srcinc:

integer array sd'field(0:sd'max'field'len);  { 0 : 14 }
byte    array sd'field'name(*)       = sd'field;
integer array sd'field'type(*)       = sd'field(8);
integer array sd'field'offset(*)     = sd'field(9);
integer array sd'field'bytelen(*)    = sd'field(10);
integer array sd'field'repeat(*)     = sd'field(11);
integer array sd'field'decplaces(*)= sd'field(12);
integer array sd'field'date'type(*)= sd'field(13);
```

Support Routines

To make our life easier, we have a standard include file with both variables and SPL/SPLash! subroutines that we use in many of our self-describing procedures:

```
sdsubr.src:

<<  Standard variables and subroutines needed to access fields in
    a self-describing file.  This file must be included after all
    variable declarations in a procedure.  >>

   integer
      file'userlabels
     ,file'foptions
     ,file'filecode
     ,current'labelnum
     ,sd'field'index
     ;

   integer array current'label(0:sd'label'len);

   subroutine file'error(local'filenum);
      value   local'filenum;
      integer local'filenum;
   begin
      xfileinfo(local'filenum);
      goto error'exit;
   end'subr;   <<file'error>>

   subroutine read'label'error(local'filenum);
      value   local'filenum;
      integer local'filenum;
   begin
      p "Unable to read label from self-describing file" err;
      file'error(local'filenum);
   end'subr;   <<read'label'error>>

   subroutine read'label(local'filenum,labelnum);
      value   local'filenum, labelnum;
      integer local'filenum, labelnum;
   begin
      blank(current'label,sd'label'len);
      freadlabel(local'filenum
                ,current'label
                ,sd'label'len
                ,labelnum
                );
      if < then
         read'label'error(local'filenum)
      else
      if > then
         if labelnum > file'userlabels then
         begin
            b'blank(outbuf,bl'outbuf);
            move outbuf := "Attempting to read label ";
            ascii(labelnum,10,outbuf'(26));
            say'errx(outbuf,35,bl'outbuf);
            read'label'error(local'filenum);
         end'if;
   end'subr;   <<read'label>>

   subroutine file'info(local'filenum);
      value   local'filenum;
      integer local'filenum;
   begin
      fgetinfo(local'filenum<<filenum          iv>>
        ,                   <<filename         ba>>
        ,file'foptions      <<foptions         l >>
        ,                   <<aoptions         l >>
        ,                   <<recsize          i >>
        ,                   <<devtype          i >>
        ,                   <<ldnum            l >>
        ,                   <<hdaddr           l >>
        ,file'filecode      <<filecode         i >>
        ,                   <<recptr           d >>
        ,                   <<eof              d >>
        ,                   <<flimit           d >>
        ,                   <<logcount         d >>
        ,                   <<physcount        d >>
        ,                   <<blksize          i >>
        ,                   <<extsize          l >>
        ,                   <<numextents       i >>
        ,file'userlabels    <<userlabels       i >>
        );
      if <> then
      begin
```

```
         p "Unable to fgetinfo on file" err;
         file'error(local'filenum);
      end'if;
   end'subr;   <<file'info>>

   logical subroutine get'field(local'filenum,offset);
      value   local'filenum, offset;
      integer local'filenum, offset;
   begin
      get'field := false;
      if sd'field'index < sd'numfields then
      begin
         if (sd'field'index mod sd'fieldsperlabel) = 0 then
         begin
            current'labelnum := current'labelnum - 1;
            read'label(local'filenum,current'labelnum);
         end'if;
         offset := (sd'field'index mod sd'fieldsperlabel) *
                   sd'entrylen;
         move sd'field := current'label(offset),(sd'entrylen);
         sd'field'index   := sd'field'index + 1;
         get'field := true;
      end'if;
   end'subr;   <<get'field>>
```

File'Error

To make our life easier we will take a simple approach to file system errors. If any MPE file system intrinsic returns an error, we call the Robelle equivalent of the printfileinfo intrinsic and then we exit the Formselfdesc procedure. Yes, we use a goto in the **file'error** subroutine. This is a good example of where a goto enhances readability and reliability.

File'Info

We have developed a standard set of subroutines for working with self-describing files. The **file'info** subroutine initializes the **file'userlabels**, **file'foptions**, and **file'filecode** variables (declared as part of the `sdsubr.src` file).

Read'Label

It is important to understand the error checking in **read'label**. MPE user labels may be allocated space, but they might not actually be written. For example, after first creating a file with user labels, none of the user labels have actually been written to the file. If we get an end-of-file condition from Freadlabel, we ignore the error unless a programming bug has caused us to attempt to read a label that is greater than the number of user labels in the file.

Get'Field

We will describe how the **get'field** subroutine works later in the section *Understanding Self-Describing Information.*

Am I A Self-describing File

We determine if a file is self-describing in two ways:

1. If the file has a filecode of 1084 and it has one or more MPE file labels.

2. The file is a KSAM file, has more than one label, we can read the last label, and the last label starts with either the string " A.00.00" or " B.00.00" (note the space at the beginning).

Here is a procedure that returns True if the passed filenum is a self-describing file:

```
$page "sd'file"
<<  Return true if the passed file is self-describing.
>>

logical procedure sd'file(filenum);
   value   filenum;
   integer filenum;
   option check 3;
begin

$include sdheader.srcinc
$include sdfield.srcinc

$include sdsubr.src

$page "sd'file/mainline"

   sd'file := false;

   file'info(filenum);

   if file'filecode = 1084 and file'userlabels <> 0 then
      sd'file := true
   else
   if file'foptions.(2:3) = 1 or
      file'foptions.(2:3) = 3 then
      if file'userlabels > 1 then
      begin   <<ksam with extra user labels>>
         read'label(filenum,file'userlabels-1);
         move sd'header := current'label,(sd'label'len);
         if sd'version = " A.00.00" or
            sd'version = " B.00.00" then
            sd'file := true;
      end'else;

error'exit:

end'proc;   <<sd'file>>
```

Creating A Self-Describing File

When we describe data structures we usually explain the input routine first and then the creation/output routine second. For self-describing files, it is easier to do it in the opposite order. We will show the structure of a simple self-describing file and then we will show the code that produced the self-describing label information for the file.

HowMessy is a Robelle program that reports on database efficiency. For years, this program has produced a report. Unfortunately, reports must be read by humans. It would make more sense for HowMessy to produce a self-describing MPE file with the efficiency information from one or more databases. You could then use a tool that understood self-describing files to report and act on the information from the file produced by HowMessy. We will show all of the routines in HowMessy's self-describing module, but first we need to know the structure of the self-describing file.

HowMessy's Loadfile

HowMessy creates a self-describing file called Loadfile. This file has one record per database/dataset/search-field for one or more databases. Here is a "form" listing of the Loadfile:

```
File: LOADFILE.GROUP.ACCT      (SD Version B.00.00)
   Entry:                     Offset
      DATABASE               X26     1
      DATASET                X16    27
      DATASETNUM             I1     43
      DATASETTYPE            X4     45
      CAPACITY               I2     49
      ENTRIES                I2     53
      LOADFACTOR             I2     57           << .2  >>
      SECONDARIES            I2     61           << .2  >>
      MAXBLOCKS              I2     65
      HIGHWATER              I2     69
      PATHSORT               X1     73
      PATHPRIMARY            X1     74
      BLOCKFACTOR            I1     75
      SEARCHFIELD            X16    77
      MAXCHAIN               I2     93
      AVECHAIN               I2     97           << .2  >>
      STDDEVIATION           I2    101           << .2  >>
      EXPECTEDBLOCKS         I2    105           << .2  >>
      AVERAGEBLOCKS          I2    109           << .2  >>
      INEFFICIENTPTRS        I2    113           << .2  >>
      ELONGATION             I2    117           << .2  >>
      FUTUREFIELDS           X136  121
Limit: 10000  EOF: 33  Entry Length: 256  Blocking: 35
```

Global Equates

To simplify programming, we use a global constant "equates" that define specific attributes of Query and Robelle self-describing files. When reading a self-describing file, we don't need most of these constants, since the necessary numbers are provided in the self-describing file header. Here are the equates that we use when creating self-describing files:

```
     sdequate.srcinc:

     equate
        sd'max'field'len       = 15
       ,sd'label'len           = 128
       ,sd'max'fieldsperlabel  = 8
       ,sd'filler'labels       = 10
       ;

     equate
        sd'date'yymmdd    = 1
       ,sd'date'ddmmyy    = 2
       ,sd'date'mmddyy    = 3
       ,sd'date'yymm      = 4
       ,sd'date'calendar  = 5
       ,sd'date'yyyymmdd  = 6
       ,sd'date'ddmmyyyy  = 7
       ,sd'date'mmddyyyy  = 8
       ,sd'date'phdate    = 9
       ,sd'date'askdate   = 10
       ;
```

Computing the Number of Labels

Before opening the Loadfile, HowMessy must determine how many labels will be needed. The following routine is used by Robelle products to compute the number of user labels for a self-describing file. Note that we continue the Query standard of reserving the first ten labels (numbered 0 to 9) for other uses:

```
$page "sd'compute'labels"
<<  Compute how many labels an SD file should have, based only
    on the number of fields.  Includes the mysterious filler
    labels.
>>

integer procedure sd'compute'labels (numfields);
   value   numfields;
   integer numfields;
   option check 3;
begin
   sd'compute'labels :=
                  (numfields-1+sd'max'fieldsperlabel) /
                  sd'max'fieldsperlabel
                  + 1 <<for the header label>>
                  + sd'filler'labels;

end'proc;   <<sd'compute'labels>>
```

Opening the Loadfile

After computing the number of user labels, we can open a new MPE file called Loadfile. We designed the HowMessy Loadfile to have records 256 bytes long. To make our life easier, we have a few global equates in the HowMessy self-describing module that we'll use throughout the rest of the examples:

```
$page "global equates and defines for the selfdesc module"

   equate
      wl'loadfile   = 128
     ,bl'loadfile   = wl'loadfile * 2
     ,bl'item'name  = 16
     ,max'field     = 22                ! fields in Loadfile
     ;
```

Here is the actual code to create the Loadfile:

```
$page "sd'open"
<<  Open the Loadfile and initialize the self-describing
    information.
>>

logical procedure sd'open(outfile,loadfile'filenum);
   integer loadfile'filenum;    ! Note by reference -- returned
   integer array outfile;
   option check 3;
begin
$include localvar.srcinc

   byte array
      loadfile'filename(0:bl'local'filename)
     ;

   move loadfile'filename := "loadfile ";

   loadfile'filenum :=
       fopen(loadfile'filename
            ,                         << foptions         lv >>
            ,1   <<write>>            << aoptions         lv >>
            ,wl'loadfile              << recsize          iv >>
            ,                         << device           ba >>
            ,                         << formmsg          ba >>
            ,sd'compute'labels(max'field)
            ,35                       << blockfactor      iv >>
            ,                         << numbuffers       iv >>
            ,10000d                   << filesize         dv >>
            ,                         << numextents       iv >>
            ,                         << initialloc       iv >>
            ,1084                     << filecode         iv >>
            );
   if loadfile'filenum = 0 then
   begin
```

```
      error(outfile,10);
      xfileinfo(loadfile'filenum);
   end'if
   else
      sd'open := true;

end'proc;   <<sd'open>>
```

Note that the filecode is 1084. For non-KSAM files, this is used to indicate a self-describing file. When you do a :Listf of such a file, MPE translates the "1084" filecode into "SD".

Writing the Self-Describing Labels

Having successfully opened a new self-describing file, it's time to write the self-describing information to the user labels. Remember that the last user label (N-1) contains the header information and the field labels are written in backward order (N-2, N-3, ...). Our routine to write the self-describing information to the Loadfile writes the field information first and then updates the header label as the last step:

```
$page "sd'write'labels"
<< Write out the labels of a self-describing file with the
   Loadfile fields.
>>

logical procedure sd'write'labels(outfile,filenum);
   value   filenum;
   integer filenum;
   integer array outfile;
   option check 3;
begin
$include localvar.srcinc
   integer
      field'index
     ,field'offset
     ,labelnum
     ;

$include sdheader.srcinc
$include sdfield.srcinc

   integer array sd'label(0:sd'label'len);

$include sdsubr.src
$page "sd'write'labels/subroutines"
   subroutine write'label(labelnum);
      value   labelnum;
      integer labelnum;
   begin
      fwritelabel(filenum,sd'label,sd'label'len,labelnum);
      if <> then
      begin
         error(outfile,12);
         file'error(filenum);
      end'if;
   end'subr;   <<write'label>>

   subroutine init'header;
   begin
      zero'buf(sd'header,sd'label'len);
      b'blank(sd'version,8);
      move sd'version   := " B.00.00";
      sd'numfields      := max'field;
      field'index       := 0;
      sd'numlabels      := sd'compute'labels(sd'numfields) -
                           sd'filler'labels;
      sd'fieldsperlabel:= sd'max'fieldsperlabel;
      sd'entrylen       := sd'max'field'len;
   end'subr;   <<init'header>>

   subroutine init'all'labels(curr'label,num'labels);
      value   curr'label, num'labels;
      integer curr'label, num'labels;
   begin
      while curr'label > num'labels do
      begin
         write'label(curr'label);
         curr'label := curr'label - 1;
      end'while;
   end'subr;     <<init'all'labels>>

   subroutine put'field(name,bytelen,decplaces,type);
      value   bytelen, type, decplaces;
      integer bytelen, decplaces, type;
      byte array name;
   begin
      zero'buf(sd'field,sd'max'field'len);
      move sd'field'name := name,(bl'item'name);
      sd'field'type      := type;
      sd'field'offset    := field'offset;
      sd'field'bytelen   := bytelen;
      sd'field'repeat    := 1;
      move sd'label(sd'entrylen*sd'field'index) := sd'field,
                                                   (sd'entrylen);
      field'offset := field'offset + sd'field'bytelen;
      sd'field'index := sd'field'index + 1;
      if sd'field'index >= sd'fieldsperlabel then
      begin
         write'label(labelnum);
         labelnum := labelnum - 1;
         sd'field'index := 0;
         zero'buf(sd'label,sd'label'len);
      end'if;
      b'blank(name,bl'item'name);
   end'subr;  <<put'field>>
$page "sd'write'labels/mainline"

   sd'write'labels := false;

   init'header;
   file'info(filenum);

   field'offset   := 0;
   sd'field'index := 0;
   zero'buf(sd'label,sd'label'len);
   init'all'labels(file'userlabels-1,sd'numlabels);

   labelnum       := file'userlabels - 2;

   sd'reclength   := bl'loadfile;

   b'blank(inbuf,bl'inbuf);

   move inbuf' := "DATABASE         "; put'field(inbuf, 26,0,1);
   move inbuf' := "DATASET          "; put'field(inbuf, 16,0,1);
   move inbuf' := "DATASETNUM       "; put'field(inbuf,  2,0,3);
   move inbuf' := "DATASETTYPE      "; put'field(inbuf,  4,0,1);
   move inbuf' := "CAPACITY         "; put'field(inbuf,  4,0,3);
   move inbuf' := "ENTRIES          "; put'field(inbuf,  4,0,3);
   move inbuf' := "LOADFACTOR       "; put'field(inbuf,  4,2,3);
   move inbuf' := "SECONDARIES      "; put'field(inbuf,  4,2,3);
   move inbuf' := "MAXBLOCKS        "; put'field(inbuf,  4,0,3);
   move inbuf' := "HIGHWATER        "; put'field(inbuf,  4,0,3);
   move inbuf' := "PATHSORT         "; put'field(inbuf,  1,0,1);
   move inbuf' := "PATHPRIMARY      "; put'field(inbuf,  1,0,1);
   move inbuf' := "BLOCKFACTOR      "; put'field(inbuf,  2,0,3);
   move inbuf' := "SEARCHFIELD      "; put'field(inbuf, 16,0,1);
   move inbuf' := "MAXCHAIN         "; put'field(inbuf,  4,0,3);
   move inbuf' := "AVECHAIN         "; put'field(inbuf,  4,2,3);
   move inbuf' := "STDDEVIATION     "; put'field(inbuf,  4,2,3);
   move inbuf' := "EXPECTEDBLOCKS   "; put'field(inbuf,  4,2,3);
   move inbuf' := "AVERAGEBLOCKS    "; put'field(inbuf,  4,2,3);
   move inbuf' := "INEFFICIENTPTRS  "; put'field(inbuf,  4,2,3);
   move inbuf' := "ELONGATION       "; put'field(inbuf,  4,2,3);
   move inbuf' := "FUTUREFIELDS     "; put'field(inbuf,136,0,1);

   if sd'field'index <> 0 then
      write'label(labelnum);

   move sd'label := sd'header,(sd'label'len);
   write'label(file'userlabels - 1);

   sd'write'labels := true;

error'exit:

end'proc;   <<sd'write'labels>>
```

Init'Header

We start our procedure by initializing most of the fields in the header label. We zero out the header label and then we fill in the variables of the header label. This file has fields with an implied decimal point, so we want to use the Robelle format of self-describing files (version number " B.00.00"). The number of fields is taken from our global equate. The number of self-describing labels is our computed number less the ten overhead labels. The number of fields in each label and the length of each field description are taken from global equates that match the values used by Query. We also initialize the **field'index** variable which is used as an index into a single label buffer (varies from 0 to sd'fieldsperlabel - 1).

File'Info

To make our code more general-purpose, we will not assume anything about the HowMessy Loadfile format (this also makes it easier to change later). Instead, we call Fgetinfo to obtain the number of labels in our file, so that we know exactly where the last label is. The **file'info** subroutine initializes the **file'userlabels** variable (declared in the `sdsubr.src` file) with the number of user labels in our file.

Init'All'Labels

To be on the safe side, we initialize all user labels in our file with binary zeroes. Note that our **write'label** subroutine uses a procedure global array called **label'buf** for writing. We initialize this buffer to binary zeroes and then continually write it out to all of the self-describing labels. We don't touch the initial ten labels reserved for other use.

Put'Field

This subroutine handles all of the details of adding a new field to our self-describing Loadfile. It initializes a new field record, moves this field record to the appropriate place in **label'buf**, and finally writes out labels as we overflow **sd'fieldsperlabel**. Each of our fields has a name (we move the name to **inbuf** and initialize **inbuf** to blanks after adding the field, a byte length, the implied number of decimal places, and a type (either byte or integer for our file). Note that the **put'field** subroutine looks after computing the byte offset of each field by incrementing a counter.

We initialize each field record with binary zeroes (just to be safe). We then fill in each portion of field information. We then move our field record to the **label'buf** at the correct offset. This works well in SPL/SPLash!, but is more of a problem in C or Pascal. In these languages, we would create a record/structure that was an array of field records and index into the structure using the current field index. As we filled up the structure, we would write out a new user label to our self-describing file.

Finishing Up

After adding all the fields, we have to see if there is one label record that has not been written to the Loadfile. If so, we write it out. Finally, the header record is written out. We do this last, since some of the variables used by **put'field** were ones from the header record.

Closing the Self-Describing File

Our next routine closes the self-describing file and handles any errors from Fclose. Pretty straight-forward MPE programming:

```
$page "sd'close"
```

```
<<  Close the loadfile and check for duplicate output files.
>>

logical procedure sd'close(outfile,loadfile'filenum);
   value   loadfile'filenum;
   integer loadfile'filenum;
   integer array outfile;
   option check 3;
begin
$include localvar.srcinc

   sd'close := false;

   fclose(loadfile'filenum,2,0);    ! Save temp
   if <> then
   begin
      error(outfile,11);
      xfileinfo(loadfile'filenum);
   end'if
   else
      sd'close := true;

end'proc;   <<sd'close>>
```

Providing a Shell

To make life easier in HowMessy, we provide one routine for the main module to call. This routine purges any exiting temporary Loadfile, creates our new Loadfile, writes out the self-describing information, and saves the Loadfile. The controlling HowMessy routine then reopens Loadfile with write-access (this may seem inefficient, but HowMessy is written in both SPL/SPLash! and HP Pascal, so it was easier to organize the code this way):

```
$page "sdcreate"
<<  Create Loadfile with all the self-describing information.  We
    purge any existing file called Loadfile, create a temporary
    one, and then fill in the labels.
>>

integer procedure sdcreate(outfile);
   integer array outfile;
   option check 3;
begin
$include localvar.srcinc
$include mpecmd.srcinc

   integer
      loadfile'filenum
     ;

   logical subroutine purge'loadfile;
   begin
      purge'loadfile := false;
      say'str "purge loadfile,temp";
      say'add rtn;
      if mpecmd'execute(outbuf,mpe'print'buffer) then
         purge'loadfile := true
      else
         error(outfile,9);
   end'subr;   <<purge'loadfile>>
$page "sdcreate/mainline"

   sdcreate := 0;

   if purge'loadfile then
      if sd'open(outfile,loadfile'filenum) then
         if sd'write'labels(outfile,loadfile'filenum) then
            if sd'close(outfile,loadfile'filenum) then
               sdcreate := 1;

end'proc;   <<sdcreate>>
```

Understanding Self-Describing Information

Our HowMessy example showed the form of the Loadfile using a format similar to the one Query uses, but the input was an MPE self-describing file instead of an IMAGE dataset. Here is our example form again:

```
File: LOADFILE.GROUP.ACCT      (SD Version B.00.00)
   Entry:                      Offset
      DATABASE             X26      1
      DATASET              X16     27
      DATASETNUM           I1      43
      DATASETTYPE          X4      45
      CAPACITY             I2      49
      ENTRIES              I2      53
      LOADFACTOR           I2      57            << .2   >>
      SECONDARIES          I2      61            << .2   >>
      MAXBLOCKS            I2      65
      HIGHWATER            I2      69
      PATHSORT             X1      73
      PATHPRIMARY          X1      74
      BLOCKFACTOR          I1      75
      SEARCHFIELD          X16     77
      MAXCHAIN             I2      93
      AVECHAIN             I2      97            << .2   >>
      STDDEVIATION         I2     101            << .2   >>
      EXPECTEDBLOCKS       I2     105            << .2   >>
      AVERAGEBLOCKS        I2     109            << .2   >>
      INEFFICIENTPTRS      I2     113            << .2   >>
      ELONGATION           I2     117            << .2   >>
      FUTUREFIELDS         X136   121
Limit: 10000  EOF: 33  Entry Length: 256  Blocking: 35
```

Formselfdesc Procedure

We have developed a stand-alone procedure for producing this output for a self-describing file. The following is the source code that we use:

```
$page "formselfdesc"
<<  If the passed filenum is a self-describing file, print a
    description of the fields in the file on $stdlist.
>>

logical procedure formselfdesc(sd'filenum);
   value   sd'filenum;
   integer sd'filenum;
   option check 3;
begin
$include localvar.srcinc
```

```
$include sdequate.srcinc

   integer
      file'code
     ,file'userlabels
     ,file'foptions
     ,current'labelnum
     ,field'index
     ,file'recsize
     ,file'blkfac
     ;
   double
      file'eof
     ,file'limit
     ;
   byte array
      filename(0:bl'local'filename)
     ;

$include sdheader.srcinc
$include sdfield.srcinc

   integer array current'label(0:sd'label'len);

   subroutine file'error;
   begin
      xfileinfo(sd'filenum);
      goto error'exit;
   end'subr;   <<file'error>>

   subroutine read'label(labelnum);
      value   labelnum;
      integer labelnum;
   begin
      freadlabel(sd'filenum,current'label,sd'label'len,labelnum);
      if <> then
      begin
         p "Unable to read label from self-describing file" err;
         file'error;
      end'if;
   end'subr;   <<read'label>>

   subroutine file'info(blksize);
      value   blksize;
      integer blksize;
   begin
      b'blank(filename,bl'local'filename);
      fgetinfo(sd'filenum      <<filenum          iv>>
        ,filename           <<filename        ba>>
        ,file'foptions      <<foptions        l >>
        ,                   <<aoptions        l >>
        ,file'recsize       <<recsize         i >>
        ,                   <<devtype         i >>
        ,                   <<ldnum           l >>
        ,                   <<hdaddr          l >>
        ,file'code          <<filecode         i >>
        ,                   <<recptr           d >>
        ,file'eof           <<eof              d >>
        ,file'limit         <<flimit           d >>
        ,                   <<logcount         d >>
        ,                   <<physcount        d >>
        ,blksize            <<blksize          i >>
        ,                   <<extsize          l >>
        ,                   <<numextents       i >>
        ,file'userlabels    <<userlabels       i >>
        );
      if <> then
      begin
         p "Unable to fgetinfo on file" err;
         file'error;
      end'if;
      if file'recsize <> 0 then
         file'blkfac := blksize / file'recsize
      else
         file'blkfac := 1;
      if file'recsize < 0 then
         file'recsize := file'recsize
      else
         file'recsize := file'recsize * 2;
   end'subr;   <<file'info>>

   logical subroutine file'is'sd;
   begin
      file'is'sd := false;
      file'info(0);
      if file'code = 1084 and file'userlabels <> 0 then
         file'is'sd := true
      else
      if file'foptions.(2:3) = 1 or file'foptions.(2:3) = 3 then
         if file'userlabels > 1 then
         begin   <<ksam with extra user labels>>
            read'label(file'userlabels-1);
            move sd'header := current'label,(sd'label'len);
            if sd'version = " A.00.00" or
               sd'version = " B.00.00" then
               file'is'sd := true;
         end'else;
   end'subr;   <<file'is'sd>>

   logical subroutine get'field(offset);
      value   offset;
      integer offset;
   begin
      get'field := false;
      if field'index < sd'numfields then
      begin
         if (field'index mod sd'fieldsperlabel) = 0 then
         begin
            current'labelnum := current'labelnum - 1;
            read'label(current'labelnum);
         end'if;
         offset := (field'index mod sd'fieldsperlabel) *
                   sd'entrylen;
         move sd'field := current'label(offset),(sd'entrylen);
         field'index   := field'index + 1;
         get'field := true;
      end'if;
   end'subr;   <<get'field>>

   subroutine print'outbuf(len);
      value   len;
      integer len;
   begin
      len := bl'outbuf;
      while len > 0 and outbuf'(len-1) = " " do
         len := len - 1;
      print(outbuf,-len,0);
   end'subr;   <<print'outbuf>>

   subroutine print'header(len);
      value   len;
      integer len;
   begin
      b'blank(outbuf,bl'outbuf);
      move outbuf'(4) := "File: ";
      move outbuf'(10) := filename,(bl'local'filename);
      len := bl'outbuf;
      while len > 0 and outbuf'(len-1) = " " do
         len := len - 1;
      len := len + 5;
      len := len + move outbuf'(len) := "(SD Version";
      len := len + move outbuf'(len) := sd'version,(8);
      len := len + move outbuf'(len) := ")";
      print'outbuf(0);
      b'blank(outbuf,bl'outbuf);
      move outbuf'(7) := "Entry:";
      move outbuf'(34) := "Offset";
      print(outbuf,-50,0);
   end'subr;   <<print'header>>

   subroutine print'trailer(len);
      value   len;
      integer len;
   begin
      b'blank(outbuf,bl'outbuf);
      len := move outbuf' := "    ";
      len := len + move outbuf'(len) := "Limit: ";
      len := len + dascii(file'limit,10,outbuf'(len));
      len := len + move outbuf'(len) := "  EOF: ";
      len := len + dascii(file'eof,10,outbuf'(len));
      len := len + move outbuf'(len) := "  Entry Length: ";
      len := len + ascii(file'recsize,10,outbuf'(len));
      len := len + move outbuf'(len) := "  Blocking: ";
      len := len + ascii(file'blkfac,10,outbuf'(len));
      print(outbuf,-len,0);
```

```
end'subr;  <<print'trailer>>

subroutine format'field'type;
begin
   if 0 <= sd'field'type <= 9 then
      case sd'field'type of begin
      <<0>> move outbuf'(31) := "?";
      <<1>> move outbuf'(31) := "X";
      <<2>> move outbuf'(31) := "?";
      <<3>> move outbuf'(31) := "I";
      <<4>> move outbuf'(31) := "R";
      <<5>> move outbuf'(31) := "P";
      <<6>> move outbuf'(31) := "J";
      <<7>> move outbuf'(31) := "K";
      <<8>> move outbuf'(31) := "Z";
      <<9>> move outbuf'(31) := "E";
      end'case
   else
      move outbuf'(31) := "?";
end'subr;   <<format'field'type>>

logical subroutine field'is'sorted(sort'index);
   value   sort'index;
   integer sort'index;
begin
   field'is'sorted := false;
   if sd'field'offset + 1 = sd'sort'keys(sort'index*3) and
      sd'field'bytelen    = sd'sort'keys(sort'index*3+1) then
         field'is'sorted := true;
end'subr;   <<field'is'sorted>>

subroutine format'sort'key(sort'index);
   value   sort'index;
   integer sort'index;
begin
   sort'index := 0;
   while sort'index < sd'sort'num'keys do
   begin
      if field'is'sorted(sort'index) then
      begin
         move outbuf'(42) := "<<Sort ";
         ascii(sort'index+1,10,outbuf'(50));
         move outbuf'(52) := ">>";
      end'if;
      sort'index := sort'index + 1;
   end'while;
end'subr;   <<format'sort'key>>

subroutine format'date'type;
begin
   if 1 <= sd'field'date'type <= 10 then
      case sd'field'date'type of begin
      <<0>> ;
      <<1>> move outbuf'(56) := "<<YYMMDD>>";
      <<2>> move outbuf'(56) := "<<DDMMYY>>";
```

```
         <<3>> move outbuf'(56) := "<<MMDDYY>>";
         <<4>> move outbuf'(56) := "<<YYMM>>";
         <<5>> move outbuf'(56) := "<<CALENDAR>>";
         <<6>> move outbuf'(56) := "<<YYYYMMDD>>";
         <<7>> move outbuf'(56) := "<<DDMMYYYY>>";
         <<8>> move outbuf'(56) := "<<MMDDYYYY>>";
         <<9>> move outbuf'(56) := "<<PHDATE>>";
         <<10>>move outbuf'(56) := "<<ASK>>";
         end'case;
   end'subr;   <<format'date'type>>

   subroutine format'decplaces;
   begin
      if sd'field'decplaces > 0 then
      begin
         move outbuf'(56) := "<< .";
         ascii(sd'field'decplaces,10,outbuf'(60));
         move outbuf'(63) := ">>";
      end'if;
   end'subr;   <<format'decplaces>>

   subroutine print'field'desc(field'repeat);
      value   field'repeat;
      integer field'repeat;
   begin
      b'blank(outbuf,bl'outbuf);
      move outbuf'(10) := sd'field'name,(16);
      if sd'version = " B.00.00" then
      begin
         field'repeat     := sd'field'repeat;
         sd'field'bytelen := sd'field'bytelen / field'repeat;
      end'if
      else
         field'repeat := 1;
      if field'repeat <> 1 then
         ascii(field'repeat,-10,outbuf'(30));
      format'field'type;
      if sd'field'type = 3 or    <<integer>>
         sd'field'type = 4 or    <<real    >>
         sd'field'type = 7 then  <<logical>>
         ascii(sd'field'bytelen/2,10,outbuf'(32))
      else
      if sd'field'type = 5 then  <<packed>>
         ascii(sd'field'bytelen*2,10,outbuf'(32))
      else
         ascii(sd'field'bytelen,10,outbuf'(32));
      ascii(sd'field'offset+1,-10,outbuf'(39));
      if sd'version = " B.00.00" then
      begin
         format'sort'key(0);
         format'date'type;
         format'decplaces;
      end'if;
      print'outbuf(0);
   end'subr;   <<print'field'desc>>
```

```
$page "formselfdesc/mainline"

   formselfdesc := false;

   if sd'filenum <> 0 then
   begin
      if file'is'sd then
      begin
         read'label(file'userlabels-1);
         move sd'header := current'label,(sd'label'len);
         current'labelnum := file'userlabels - 1;
         print'header(0);
         field'index      := 0;
         while get'field(0) do
            print'field'desc(0);
         print'trailer(0);
         formselfdesc := true;
      end'if;
   end'if;

error'exit:

end'proc;   <<formselfdesc>>
```

A Different Logical Structure

Our HowMessy/Loadfile example had a number of separate procedures. Our Formselfdesc procedure is self-contained, but we will describe each subroutine in this procedure.

Formselfdesc Variables

We include our standard files for the global self-describing equates, header layout, and field layout. We also have a number of local variables that are used for indexing through the field labels and other variables needed to enhance the output listing (e.g., the number of records in the self-describing file).

File'Is'Sd

This is our standard **sd'file** procedure, rewritten to work as a stand-alone SPL subroutine. **File'Is'Sd** looks after calling the **file'info** subroutine which calls Fgetinfo. We initialize a number of variables during the **file'info** call. Some of these are used for obtaining self-describing information and some are used to enhance the format of our form output (e.g., the filename and the file limit).

Read'Label

The basic strategy we use in this routine is to read a specific label into a buffer called **current'label**. We then move this label to the appropriate self-describing header or field buffer. **Read'label** is careful to check for file system errors and abort if it finds any.

Get'Field

This subroutine is the key to understanding self-describing files. When **get'field** is called the last label of the file has been read into the **sd'header** record. The variable **field'index** is initialized to zero is used as a counter of self-describing files. Each call to **get'field** returns one field description in the **sd'field** record.

When first called, **current'labelnum** contains the number of the last label (minus one, since MPE numbers labels starting at zero). We check to see if we need to read in a new label with the statement:

```
      if (field'index mod sd'fieldsperlabel) = 0 then
```

Note that we use **sd'fieldsperlabel** as the divisor. This is the value from our **sd'header** record and not our equate that we use when creating self-describing files. **Get'Field** assumes that the current label record is in the buffer **current'label**.

Each user label contains one or more field descriptions (in most cases there are eight per label). We compute an offset in the label where the current field description is and then we move the field description from **current'label** to our **sd'field** record.

Print'Field'Desc

This routine looks after printing out the description of one field. We use the same routine whether we are dealing with Query (" A.00.00") or Robelle (" B.00.00") self-describing files. We do have to adjust the byte length for " B.00.00" self-describing fields, so that the output looks similar to what Query would produce for an IMAGE dataset. Note how the **format'type** routine handles IEEE floating point for either type of self-describing file.

For " B.00.00" self-describing files, we can produce extra information. This is handled by the **format'sort'key**, **format'date'type**, and **format'decplaces** routines (which are only called for " B.00.00" self-describing files.

Format'Sort'Key

The sort information is stored in the **sd'header** record as an offset, a length, and a type. There is no direct way for us to tell that a field is sorted. Instead, we index through all of the sort keys checking if the sort key matches the current field definition (there might not be a match). We use the index into the sort information as our key to print for the user.

Field'Is'Sorted

To make our code clearer, we encapsulate the code for checking if a specific sort key matches the current field in a subroutine. By giving this subroutine a descriptive name, we make the intent of the **format'sort'key** routine clearer. Our **field'is'sorted** routine checks that the offset (adjusted appropriately for one-based and zero-based offsets) and the byte length of the field and the sort key match. We decided to ignore the data type (the sd'type and the sort'type have different values).

Summary

It's harder to understand self-describing files than it is to create them. When creating self-describing files you often only use a few of the self-describing features, but when understanding them there are no features that you can leave out.

KSAM Self-Describing Files

Self-describing KSAM files are a little trickier to deal with. The 1084 filecode used for self-describing MPE files doesn't work well for KSAM. It is more difficult to create a new KSAM file, since all of the key information must be passed to Fopen. Here are a few hints for creating and understanding self-describing KSAM files.

SD (1084) Filecode

You can create a KSAM file with a filecode of 1084, but the resulting :listf,2 gives no hint that the file is a KSAM file. Here's an example Build Command of a compatibility-mode KSAM file with a filecode of 1084 and the resulting :listf,2.

```
:run ksamutil.pub.sys
>build file1;rec=-80,16,f,ascii;keyfile=file1key; &
            key=i,6,2;code=1084
>exit
:listf file1@,2

FILENAME  CODE  ----------LOGICAL RECORD---------  ----SPACE----
                  SIZE  TYP       EOF    LIMIT R/B  SECTORS X MX

FILE1     SD      80B  FA          0     1023  16       48  1  *
FILE1KEY  KSAMK  128W  FB         98       98   1      112  1  8
```

Notice how there is no way to identify `file1` as being a KSAM file. For this reason, we don't use the 1084 filecode on self-describing KSAM files.

Creating KSAM Self-describing Files

We use three steps to create self-describing KSAM files:

1. Compute the number of labels (used in the KSAMUTIL or MPE/iX build command). You could use our **sd'compute'labels** subroutine or you can compute the number of labels as the truncated value of:

   ```
   labels = (#fields + 7) / 8 + 11
   ```

2. Build your KSAM/V file with KSAMUTIL and specify Labels=[the number computed above]. For KSAM/XL, use the Build Command with the userlabel keyword ;ULABEL=*x* (where *x* is the number computed above).

3. Fopen the file as an old file with write access.

The most difficult part is computing the number of labels. For example, if we have eight fields:

```
    Labels = (8 + 7) / 8 + 11
           = 12
```

MPE V/E:

```
    :run ksamutil.pub.sys
    >build file2;rec=-80,16,f,ascii;keyfile=file2k; &
     key=i,6,2,,duplicate;labels=12
    >exit
```

MPE/iX:

```
    :build file2;rec=-80,16,f,ascii;key=(i,6,2,dup); &
                 ksamxl;ulabel=12
```

Understanding KSAM Self-Describing Files

Our **sd'file** routine returns true if a given file is self-describing. If you examine the code in this routine carefully, you'll see that for KSAM files we have the statements:

```
    if file'foptions.(2:3) = 1 or file'foptions.(2:3) = 3 then
       if file'userlabels > 1 then
```

Note that we check for more than one user label. Why don't we check for more than zero user labels? All self-describing files must have at least two labels (one for the header information and one or more for the field information). When we first implemented our **sd'file** routine we only checked for more than zero user labels. What we found was that many users had accidentally built KSAM files with one user label (which was almost always empty). We have no idea why this seemed to be so common, but by checking for at least two labels we eliminated a lot of KSAM files that were not self-describing.

Future Self-Describing Formats

We were motivated to create the new Robelle format self-describing files in order to provide a better interface between our product Suprtool and ASKPlus from ARES of France. Pierre Senant of ARES is the R&D Manager and the two of us worked out the " B.00.00" self-describing format (actually we forced most of the format on poor Pierre).

ARES have been doing significant R&D work on UNIX and a portable version of ASKPlus. As an example of how far we can go with self-describing information, here is an extract of Pierre's design for a UNIX implementation of self-describing files.

SDASK Files

A C/ISAM file is composed of two files, a data file and an index file. An SDASK file defines another file called a 'label file' which contains the complete description of the data file. Like MPE self-describing files, an SDASK file is composed of a header portion and a description of each field. The data file and the label file must be located in the same directory.

Header Format

Pierre's header contains a lot more information than our MPE header label. Here are the parts of the header:

- Version number.
- File code (1085).
- Checksum (currently unused).
- Number of fields per record.
- Record length (in bytes).
- Number of records.
- Number of sort keys.
- Password.
- Total field area length (in the SDASK file).
- File type (flat, C/ISAM, KSAM, Unibol, ...).
- Data file name.

- Unibol area (for migration from IBM/36 to UNIX)
- Filter: logical expression defining a condition that must be True for the entries taken into account. Originally developed for Unibol files, but this feature can be used for any other system.

Field Description

Each field description is variable length:

- Field type (U, X, I, J, K, P, R). Additional information for Ascii fields are: Roman-8, PC-8, ANSI-8, Mac-Apple, EBCIDC, and ISO7-1 ... ISO7-13. For Integer fields, there is additional information for Intel versus HP. For Real fields, there is additional information for IEEE versus Classic.
- Length (in bytes).
- Offset.
- Scale (number of decimal places).
- Repeat factor.
- Flags:
  - Null value allowed. If this flag is True, each entry in the data file is preceded by a bitmap field. Each bit indicates whether the corresponding field value is Null or not.
  - Hidden field.
  - Key.
  - Duplicate key allowed.
- Field name length.
- Field name.
- Title length.
- Title.
- Edit mask length.
- Edit mask.
- Key file name length.
- Key file (reserved for future implementation on MS-Dos).

Sort Information

Sort descriptors are also variable length:

- Expression length.
- Sort expression (in ASKPlus syntax). For example,
```
        cust-name

        cust-zipcode cat cust-address
```
- Flag: ascending/descending.

Conclusion

Self-describing files are a great idea. As users, we almost always create MPE and KSAM files with a fixed record structure in mind. By default, this record structure is lost when we build a file. With self-describing files, we can retain the structure of our files.

A Final Example

The HP 3000 has a rich set of tools based on IMAGE. One reason that so many good tools could be written for IMAGE was the DBINFO intrinsic. This intrinsic let any program discover the structure of an IMAGE database. Self-describing files provide the same flexibility for MPE and KSAM files.

In this example, we show how two tools can be combined by using self-describing files. Our HowMessy program reports on database efficiency. While doing so it creates a self-describing file with the statistics for a database. Once you have this file, it's possible to use Suprtool to check for certain boundary cases. For example,

```
:run howmessy.pub.robelle            {create "loadfile"}

Enter database:  test.suprtest
```

HowMessy creates the self-describing file called Loadfile (with the structure that we've shown previously). We now use Suprtool to create a file that has all detail datasets that are more than 85% full that also have a capacity greater than one:

```
:run suprtool.pub.robelle
>input   loadfile
>if      datasettype = "D"  and   &   {detail dataset}
         capacity    > 1    and   &
         loadfactor  > 85.00          {more than 85% full}
>output  loaddetl,link                {create SD file}
>exit
```

At Robelle, we would use our Xpress electronic mail system to mail the `loaddetl` file to the system manager. Another alternative would be to to extract the database and dataset names and use them to create a batch job to automatically increase the capacity of detail datasets more than 85% full. The possibilities are endless, but only because HowMessy could provide information to Suprtool via the self-describing file.

Software Tools

Few software tools are capable of creating or understanding self-describing files. This is a shame, since self-describing files are a powerful data structure. One reason that so few tools handle self-describing files is that documentation on self-describing files has been non-existent. I hope that by publishing this description and the programming examples in this paper that more vendors and users start creating and accepting self-describing files.

PAPER NO.:	3063
TITLE:	Integrating Voice Response Into An HP Environment
AUTHOR:	Will Strother Comuter Communications Specialists 6529 Jimmy Carter Blvd. Norcross, GA 20071 (404) 441-3114

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

Paper 3064

Using Terminal Emulation to Perform OLTP Benchmarks

Gary A. Showers, CDP

State Farm Insurance Companies
One State Farm Plaza
Bloomington, IL 61710
(309) 766-7742

Abstract

Large, distributed on-line transaction processing (OLTP) environments require routine and pro-active performance management to help assure success. Benchmark and performance tests of software and hardware should play an important role in any performance management strategy. The results of these tests can be used as input into capacity planning predictions, to determine the consequences of CPU and other hardware changes, and to measure the impact of new or updated software on a production environment—all before making any changes to production systems. This paper presents State Farm corporate data processing strategies and techniques to perform OLTP benchmark and performance tests. The paper's focus is the description, rationale, benefits, strengths, and weaknesses of terminal emulation using a terminal emulation product written for State Farm by Hewlett-Packard called Wrangler.

Introduction

Performance evaluation is of interest to computer system administrators, end-users and system designers. System administrators and designers want to provide the highest performance and best response time at the lowest cost. End-users want to obtain the best response time and highest performance at the lowest cost (2, pg. 1). Terminal emulation is one of several performance evaluation tools available to help meet this high performance/low cost goal.

This paper examines terminal emulation's role in an active performance management strategy. It is not a "how to" guide; it is a "what, why, and when" profiie of using terminal emulation. An introduction to test types and terminal emulation provides the necessary background. The next section describes the Wrangler terminal emulation product. The remainder of the paper discusses the rationale behind performance testing and reviews State Farm's experiences using Wrangler terminal emulation to perform OLTP benchmark and performance tests.

Test Classifications

There are six types of computer system tests: benchmark, capacity planning, function, performance, regression, and stress. All of these tests play an important role in a well-designed performance management plan. State Farm routinely uses terminal emulation to perform benchmark, capacity planning, and performance tests. Stress tests are also executed under special circumstances. State Farm does not currently use terminal emulation for function testing or regression testing.

Benchmark and Capacity Planning Tests

Benchmark tests compare specific performance characteristics of two or more systems by running standard workloads on the systems. A close relative of the benchmark test is the capacity planning test. Capacity planning tests compare specific performance characteristics of systems by adding or changing a system's CPU, memory, disk, and other resources. Benchmark and capacity planning tests help to determine which system configuration will meet current and future processing needs. These tests also identify the performance effect of system configuration changes. Throughout this paper the term benchmark refers to both benchmark and capacity planning tests.

Function Tests

Function tests verify that each new function of an application works as intended. The test cases should incorporate all functional possibilities, including errors.

Performance Tests

Performance tests determine the relative performance of application software or system software. The usual technique is to repeatedly tune and measure an application or system to quantify and improve its performance. Another technique is to compare the performance characteristics of a baseline environment to a modified environment. The latter technique does not require a tune and measure cycle unless the results uncover performance problems.

Regression Tests

Regression tests verify that old application functions still operate correctly after adding or modifying application functions, or making system software configuration changes.

Stress Tests

Stress tests reveal problems in application and system resource utilization by driving the system to extremely high transaction rates. Tests that use low or average workloads may not expose resource utilization problems.

What is Terminal Emulation?

Testing and measuring the performance of a computer system requires a way of putting workloads on the system. Terminal emulation provides a load on computer systems without relying on real terminals or terminal users. A terminal emulator simulates terminals and users communicating with applications and the operating system on a computer system under test (SUT). Terminal emulation drives applications so that they operate as they would under actual conditions. The key qualities of terminal emulation are providing representative, repeatable, and controlled transaction workloads.

Terminal Emulation Components

A terminal emulator customarily runs on a dedicated computer system called the driver system. The driver system connects to the SUT through telecommunications ports, one port per simulated terminal. The number of terminal ports necessary depends on the test objectives, varying from one port to hundreds of ports. The terminal emulator reads script files containing each terminal's application and system commands. These commands are sent to the SUT through the terminal ports at specified intervals (see Figure 1).

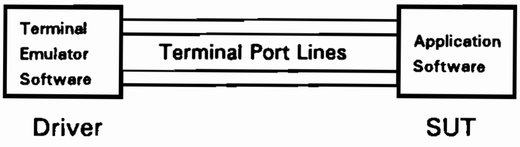


Figure 1. Terminal emulation driver and SUT configuration.

Terminal emulation progresses in three phases: preemulation, emulation, and postemulation. Terminal emulators consist of three distinct components used in these phases. The terminal emulator's preemulation phase component defines the application and system workload scripts. The emulation phase component uses the scripts to run the test and collect transaction data. The postemulation phase component reduces the collected data and produces reports for analysis (2, pg. 133).

Terminal Emulation Phases

Preemulation Phase

The preemulation phase defines the specific characteristics of the terminals, users, and programs to use during a test. Example characteristics include: terminal line speeds, terminal communication protocols, user think times, user data entry rates, application and system command definitions, command sequences, and the programs to use (2, pg. 134). A terminal emulator generally provides control over how the user scripts execute during the test. Examples of terminal emulator script control parameters include: script processing logic, user think times, script start and end times, and script processing repetition counts (2, pp. 135).

Emulation Phase

The emulation phase uses a terminal emulator to perform the test. The test duration will vary according to the objectives of the test, varying from a few minutes to several hours. The test operator can observe the number of active scripts, the number of characters sent and received, and the elapsed time for each script. It is also possible to monitor single or multiple scripts to observe the actual terminal and CPU data ordinarily presented to the terminal user (2, pg. 136). The terminal emulator provides script control, such as pausing, resuming, and stopping individual or multiple scripts. The terminal emulator also records a detailed log of script activity including transaction time stamps.

Postemulation Phase

The postemulation phase uses a data reduction program to process the log file produced in the emulation phase. Data reduction is a CPU and I/O intensive process. Transferring the log file to a more powerful system will expedite the reduction. An added advantage of moving the log file off the driver system is the ability to concurrently conduct new tests and reduce previously collected test data. The terminal emulator's data reduction program reports a chronological trace of all script activity. The data reduction program generally supports statistical analysis, such as average and standard deviation response time calculations, and it may support data export facilities to provide input into other analysis tools (2, pg. 136).

Wrangler History

Wrangler started out as an HP internal terminal emulation tool called Terminal Emulator and Performance Evaluator (TEPE). TEPE was originally developed to run on HP1000s and Series II and III HP3000s (1, pg. 1-1). Eventually TEPE was unable to provide the services required to test HP's newer MPE V/E systems. To address these deficiencies, the HP Performance Technology Center replaced TEPE with a new HP internal terminal emulation tool called Wrangler.

State Farm corporate data processing used TEPE and Wrangler to conduct benchmark and performance tests. Because both tools were HP internal tools State Farm received no official support. HP's introduction of MPE XL hardware and software again outgrew Wrangler. The HP support and MPE XL issues prompted State Farm to commission HP to write a new version of Wrangler that included MPE XL functionality, full HP support, and product enhancements. State Farm uses the new version of Wrangler to perform all HP benchmark and performance testing.

Wrangler Features

The Wrangler product feature set includes the following (1, pg. 1-3):

* A powerful, extensible command syntax providing features such as script declaration, terminal setup, process control, status monitoring, and MPE commands.

* The ability to drive up to 200 simulated terminal ports per Wrangler session while incurring low process overhead on the driver system.

* Support of multiple driver systems, multiple MPE V/E and MPE XL SUTs, and multiple Wrangler sessions on one driver system.

* The driver system currently must be a MPE V/E system; a MPE XL Wrangler driver is unavailable.

* Low overhead exception checking and status display of the ongoing test.

* A script generation system that allows for the capture and conversion, on both MPE V/E and MPE XL systems, of actual terminal I/O into scripts for replay later.

* Powerful and flexible data reduction capabilities including script command tracing, response time calculation, and logical transaction definition.

Wrangler's features add up to a powerful, but complex product that requires hands-on experience and trial and error to use successfully. Don't be afraid to use Wrangler! Its benefits far outweigh its learning curve. There is only one significant hurdle to overcome before it is possible take complete advantage of Wrangler. The hurdle is the complex and cryptic nature of Wrangler script source code.

Wrangler scripts reproduce the actual I/O transactions sent to and from a terminal and a computer system. These transactions include what the terminal user types, as well as all handshaking and protocol transactions to and from the terminal and the CPU. The scripts' complexity arises from these low level transactions and how they relate to and depend on the content and sequence of other transactions. Using Wrangler successfully requires the ability to read, understand, and modify these transactions.

Wrangler Structure

Wrangler has components designed for use in all three phases of terminal emulation. A description of Wrangler's primary components (1, pp. 1-3, 1-5) is next, followed by Figure 2 (1, pg. 1-4), illustrating the components and their communication paths.

Wrangler Preemulation Phase Components

ROUNDUP is a utility that captures live terminal I/O transactions directly from the MPE file system and writes this information to a log file.

MAKETEPE (not shown in Figure 2) is a utility program that processes the Roundup log file and creates an individual TEPE ASCII command file for each terminal monitored.

TEPE ASCII COMMAND FILES contain the captured terminal I/O transactions, one file per terminal. If script customization is necessary, any text editor can modify the TEPE ASCII command files.

TEPECOMP is a utility that compiles a TEPE ASCII command file and produces a special binary script file formatted for input into a TEPE process.

Wrangler Emulation Phase Components

CHECKER is a Wrangler son process that uses the data passed from the SUT to drive a status display showing how the test is progressing. Checker logs and optionally displays the data being passed between the driver and the SUT.

SCRIPT FILES are binary files containing the application and system commands for the TEPE processes. Each TEPE process can use its own unique script file or multiple TEPE processes can share a single script file.

SUT PORTs are terminal ports on the driver system connected to terminal ports on the SUT. A Wrangler script appears to the SUT as a terminal executing application and system commands.

TEPEs are Wrangler son processes that read and interpret the script files while communicating with the SUT. There is one TEPE process per SUT port.

WRANGLER is the user interface and master process that controls the actions of the other processes in the Wrangler system. The Wrangler system uses temporary message files for process to process communication.

WRANGLER XEQ COMMANDS FILE contains Wrangler commands specifying a test's script files and run-time parameters.

Wrangler Postemulation Phase Component

TEPEDRP is a utility that produces formatted reports of the script activity recorded in the Checker log file. TEPEDRP has options to trace script activity, logically define transactions, and calculate response times.

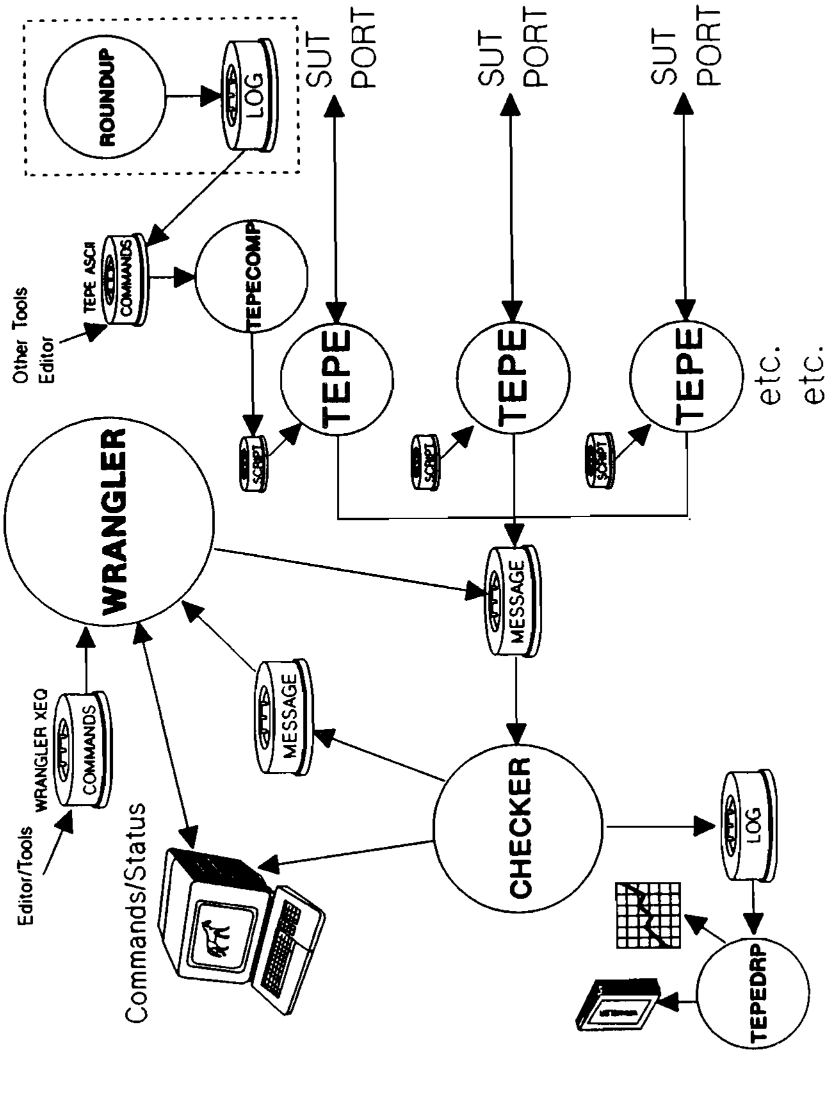


Figure 2. Wrangler System Structure.

To Test or Not to Test

The main objective of testing is to find performance problems before putting an application or system into production. It is less expensive to find and fix a problem sooner than later. Today's businesses demand a high degree of computer system reliability, availability, and performance. Benchmark and performance tests of software and hardware must play an important role in any performance management strategy. State Farm has made a 100% commitment to testing. The large number of applications running on numerous systems makes our exposure to problems and the impact of problems too great to ignore or address less than completely.

The costs of not testing or testing inadequately may include:

* Unacceptable application and system response times.
* End-user dissatisfaction
* Inefficient or inappropriate application and system resource utilization.
* Potential loss of business.
* More system outages.
* More personnel for production system maintenance.
* Slower software migration.
* Less new development.
* Lost or incorrect data.
* Higher data processing costs.
* Missed business opportunities.

Setting up and maintaining a proper test environment that uses terminal emulation requires a significant effort and many resources. State Farm has dedicated several MPE V/E and MPE XL computer systems, application development analysts, and support analysts to performance testing and terminal emulation. Planning and executing tests requires the cooperation and coordination of all these resources. Well-organized testing should be coordinated by a performance testing team containing members from the appropriate areas in the data processing and user liaison departments.

Benchmark and performance testing must be an integral step in every important application or system development project. Review all projects, especially small and apparently insignificant ones, for their potential to be performance tested. Total management support of testing is crucial to establishing and maintaining a successful performance testing strategy.

Discussed below are several useful examples of actual State Farm test cases illustrating the benefits and flexibility of using Wrangler for benchmark and performance testing.

* Capacity planning, benchmark and stress tests of MPE V/E and MPE XL systems. These tests provided valuable system sizing information tailored to State Farm's current and future application workloads. The results were substantial cost savings gained by acquiring the proper MPE XL systems to replace State Farm's MPE V/E systems.

* Performance testing all new applications. There have been several tests that identified a detrimental performance impact on application response time or system resource utilization. By using a measure and tune cycle, we corrected the performance problems before putting the new applications into production. The results are satisfied customers, attained by providing appropriate system capacity, excellent user response times, and continued user productivity.

* A performance test combining HP system and IBM system processing workloads. State Farm's claim processing environment relies on HP and IBM system information and transaction processing. HP and IBM cooperative claim processing will continue to increase over time. The test solution used Wrangler to send an HP system originated transaction workload to an IBM IMS system. IBM's TPNS terminal emulator, running in conjunction with Wrangler, provided a comprehensive IMS workload. This test illustrates one of many uses for Wrangler. It also provided more representative, accurate, and reliable workloads for testing State Farm's HP and IBM systems. The results are high quality, efficient applications and systems for our users and our customers.

Summary

Performance is central to the design, acquisition, and use of computer systems. The goal of computer systems' management, analysts, and users is to get the highest performance and best response time for a given cost (2, pg. 3). The costs of not testing applications and systems far outweigh the costs of testing.

Terminal emulation enhances all types of testing by providing a flexible, controlled system to supply a workload on computer systems without requiring real terminals or terminal users. Setting up and maintaining a proper test environment that uses terminal emulation requires a significant effort, many resources, and total management support. A well-designed terminal emulation test strategy provides the means to determine the impacts of a variety of alternative configurations and workloads, to tune applications and systems, to test the new functionality of applications, and to verify that old application functions still work correctly.

State Farm uses the Wrangler product to perform benchmark, capacity planning, performance, and stress tests. Wrangler is a powerful and flexible terminal emulation package that provides representative, repeatable, and controlled transaction workloads. The learning curve for Wrangler can be steep, but the rewards are great. State Farm has used the latest version of Wrangler since July 1990. During that time, State Farm has performed hundreds of invaluable benchmark tests, performance tests, and stress tests on the behalf of dozens of projects.

References

1. Hewlett-Packard Company, Wrangler, product manual, Hewlett-Packard Company, 1991.

2. Raj Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, John Wiley & Sons, Inc., New York, 1991.

The Warning Signs
A Pop Quiz on Quality

By Robert Green and David Greer

Robelle Consulting Ltd.
Unit 201, 15399-102A Ave.
Surrey, B.C. Canada V3R 7K1
Phone: (604) 582-1700
Fax: (604) 582-1799

Abstract

Everyone is in favor of software quality, but not everyone is producing quality software. How can you tell if a software group has gone off the track? It could be your DP department, your computer manufacturer, or even one of your software suppliers. The ideal software group uses feedback and repeated development cycles to find out what users really need. But it also uses rigorous software engineering. Despite the constant changes, the ideal software group ensures that existing features continue to work and future changes are possible. We have organized our ideas into a non-threatening Pop Quiz, consisting of tell-tale phrases that you may recognize, phrases that warn of a troubled software project, phrases such as "that's not my job" and "it's against our policy."

PAPER #3101

The Warning Signs
A Pop Quiz on Quality

By Robert Green and David Greer

Robelle Consulting Ltd.

Everyone is in favor of software quality, but not everyone is producing quality software. How can you tell if a software group has gone off the track? It could be your DP department, your computer manufacturer, or even one of your software suppliers.

Introduction by Robert Green

I was presenting my paper, **Improving Software Quality**, before a hometown crowd in Vancouver, when a local consultant rose to ask a question. "Responding to the users sounds fine," she said, "but I've seen a lot of systems where the programmers did constant changes and patches to give the users whatever they wanted. Most of the systems were a buggy, kludgy, impossible mess. How do you avoid that?"

"That's the topic for another paper," I replied, with a quick tap dance to distract the audience. Later, I admitted to myself what a good point she had made.

Robelle creates software tools. About once a month we release a new version of Qedit and Suprtool. We are constantly on the go, so how do we avoid producing a disaster?

I went to my partner David Greer for the answer. At Robelle I'm the one who leans toward wild, creative impulses. David is stronger on reliability, discipline, and long-range thinking. I was so concerned by bureaucratic software groups which didn't satisfy user needs that I was overlooking another style of shop: groups with weak software skills who react to user complaints as fire fighting.

The ideal software group uses feedback and repeated development cycles to find out what users really need. But it also uses rigorous software engineering. Despite the constant changes, the ideal software group ensures that existing features continue to work and future changes are possible. My previous paper on improving software quality was only half the story. Now David Greer and I finish it off.

Teaching Quality Without "Should" and "Ought"

To avoid having our paper degenerate into a sermon, we have organized our ideas into a non-threatening Pop Quiz -- a way to quietly rate your quality record. The quiz consists of giveaway phrases that warn of a troubled software project, remarks such as "that's not my job" and "it's against our policy."

See how many of these phrases you've heard. Score 5 penalty points for each one you hear regularly. We made the penalty 5 points so you could fudge if you want to.

Disclaimer: Despite the fact that we use Hewlett-Packard for some examples, we are not out to get HP. We mention HP because we know them and you know them, not because we think their software quality is substandard. To show it isn't personal, we've thrown in a few Robelle follies too. All shops are guilty of some of these quality slips from time to time.

o o o

Ready? Here's our first warning sign, a classic programmer excuse:

That's a feature, not a bug.

If the user then points out the spot in the documentation showing that he is right, there is still the second tell-tale excuse:

That's an error in the manual.

With those two replies, a programmer can deflect any conceivable bug report.

The longest-running problem report at Robelle is caused by a "feature" of MPE. A batch job can log onto another MPE system by doing a Remote Hello Command and creating a remote session. But, if the job runs Qedit (or any program) on the remote system, MPE tells Qedit it is in an interactive session, not a batch job. Qedit cannot tell this phoney session from a true session.

Qedit attempts to do a CRT status request, assuming that the session comes with a person and a terminal. Because no one is there running the sesion, the first Qedit command gets read as the CRT status! The DS/3000 Manual **from 1978** is aware of this problem and warns about it, but in 13 years HP has not fixed it. It's a feature, not a bug.

Reluctance to admit problems leads to a reputation as a "Black Hole". Questions go in, but nothing useful ever comes out.

o o o

When users gather, they swap "war" stories. One user was overhead at lunch, complaining about his software vendor. When he called the support line, their first question always seemed to be:

Why would you want to do that?

This statement suggests a superior attitude on the part of the programmer, an attitude that is seldom backed by reality. Customers do find unexpected and incredible ways to apply programs to solve their problems. The sign of a truly great program is that even when used in unanticipated ways, it still works.

Whether a program is useful is up to the customer to judge. Take an example from the carpet business, where despite superior technical knowledge at the factory, the customer knew best:

> One of our customers in Europe came to us several years ago with his own testing spec for carpet foam backing. We were a bit put out that someone thought they could test it better than we could. We told him not to worry. Dow measures for foam stability, molecular weight distribution, particle size conformity, percent of unreacted monomer, adhesion strength -- all the vital things. We told him, "You're going to get the best there is, real quality!"
>
> Well, three times we tried to ship him the order, and three times he sent it back. Now, that gets annoying. So we asked him, "What's the deal?" And he told us, "Your product can't pass my roll-stool test!"... What he did was take the bottom half of an office chair, put a weight on it, and spin it around on a piece of test carpet 30,000 times... If the carpet sample didn't delaminate from the foam, you passed the test and got the order... Quality is what the customer says he needs, not what our tests indicate is satisfactory. [I. Snyder of Dow, in T. Peters]

o o o

If there are no questions, everyone must be happy.

At a Management Roundtable where the users submitted only 18 questions, the moderator jokingly remarked, "This proves the customers are happy."

Wrong.

Customers who don't voice their complaints, when given a chance, have given up expecting answers. The unconscious attitude behind this warning sign is that customer complaints are an irritant to be tolerated.

Real user complaints are good, not bad. People who use and like software constantly think of more problems the software could solve. This shows up as increased "complaints". If you actually fix something for them, watch out! The complaints will escalate dramatically.

At Robelle, we have a saying: If we send out a batch of new software and **no one complains**, there is only one conclusion.

It means that **no one tried it.**

o o o

We've lost the source code.

On his first outside consulting job, Bob Green found that the client had lost his source code.

The client's billing program was taking 2 days to run on a sampling of accounts--to do all the accounts it would be running continually. The system was designed by an expensive consulting firm, then programmed by contractors. The client had only a junior programmer to make the inevitable patches and fixes. There were lots of source files, but no one knew which ones compiled into the current version of the billing program. It took most of a day to find the proper source files and get them to re-compile. After that it took only an hour to fix the program.

Programs are valuable assets that depend upon an infrastructure for their preservation. If you take shortcuts in development, you will lose control of this asset.

What is needed is simple, but requires discipline. For every program, there is a job stream that recompiles the current version. This job stream shows the source files that go into the program and how to combine them. The test environment is kept separate from production. Another standard job moves a new version from testing into production.

o o o

We're too busy to document that.

Having each programmer in a shop re-invent the wheel is inefficient. Software techniques uncovered by a software group can be a valuable asset if they are documented. There is an even better way of retaining knowledge. Use **libraries of code**. Reusable code leverages the investment. One programmer suffers to solve a problem so the others won't have to suffer as well.

As an example, consider double-sided printing on the LaserJet. In our shop, Dave Lo first added this feature to our Prose text formatter by hard-coding the Escape sequence into the program. Then Bob Green hard-coded it into Qedit. When David Greer wanted to add double-sided printing to Suprtool, he asked, "Why is the Escape sequence hard-coded in Prose and Qedit, when we have a library routine that could have held that knowledge?"

David added the Escape sequence to our library code and removed it from the individual programs. Now we can add double-sided printing to the rest of our programs without having to re-learn the Escape sequence.

> The good programmer writes software that can be reused, even if he doesn't see a reuse immediately on the horizon. His experience ... assures him that someone from the next hallway, reasonably soon, will be asking, "Do you have a module that ... ." He also knows that writing reusably will force him to define clean interfaces. [D. Boundy]

o o o

When a programmer can't find the reason for a particularly bizarre and puzzling bug, there is one sure fire excuse.

It must be a hardware problem.

In all the bugs we have investigated in 13 years in business, only one turned out to be a true hardware problem.

Our Qedit program was repeatedly losing track of lines in a file at one customer site, but we couldn't repeat the problem on our machine. Through painful hours spent in Debug, we finally proved that the Load Instruction was failing on this one customer's computer, but only when indexing was done without indirection. And Qedit was the only program on the system that used this unusual mode of the Load Instruction.

The ugly truth is that hardware is amazingly reliable and software is not.

o o o

It would cost too much to update the paperwork.

The late Dr. Richard Feynman of Cal Tech did a famous study of the Challenger shuttle disaster. He rangea widely throughout NASA and its contractors, talking to anyone who could shed light on the quality problems.

A group of production workers had found a simple way to improve the calibration of the rocket engines, but it was never implemented.

> The foreman said he wrote a memo with this suggestion to his superiors two years ago, but nothing had happened yet. When he asked why, he was told the suggestion was too expensive. "Too expensive to paint four little lines?" I said in disbelief. They all laughed, "It's not the paint; it's the paperwork. They would have to revise all the manuals."
>
> The assembly workers had other observations and suggestions... I got the impression they were very interested in what they were doing, but they weren't being given much encouragement. Nobody was paying much attention to them. It was remarkable that their morale was as high as it was under the circumstances. [R. Feynman]

o o o

When people want to cut corners, you hear assertions like this:

No one will ever notice.

What they mean is that the users are too stupid to recognize quality when they see it.

At Ford Motor Company, they once came up with a scheme called PIP, Profit Improvement Program.

> The purpose of PIPs [Profit Improvement Programs at Ford] was to bring down the costs of making a car by taking them out of an existing budget; an example might be the decision to equip a Mercury with Ford upholstery, which was cheaper. Some traditionalists were convinced that the PIPs systematically reduced quality, that it was automotive sleight of hand, and that the covert philosophy behind the program was that the customer would never know the difference. PIPs quickly became part of the vernacular, turning into a verb. "What happened to that hood ornament?" "Oh, it got pipped." [D. Halberstam]

You seldom know which features will be important to your users. Success demands attention to all details. In software, users notice the little things. They seem to be more sensitive to details than to the big picture, perhaps because they take the overall objective for granted but the details drive them crazy day after day. The one enhancement to Qedit which received the most positive feedback from the users was a tiny and simple change: allowing the entry of MPE commands without the preceding colon.

o o o

One more Go To won't hurt much.

Good programmers don't use Go To to solve their logic problems. They structure their code as easily as they breathe. They keep their programs within their intellectual grasp by using limited control structures: While, Do-Until, If-Then_Else, and Case. Keeping it simple keeps it easy to understand. The problem with Go Tos is that you can build convoluted structures with them.

Well-structured programmers limit the scope of their data structures. If a variable is only needed within a procedure, they make it a local variable not a global variable. If a procedure needs to access a global they either pass it in as a parameter or export it. If the programming language allows, they distinguish parameters which are input only from those that are both input and output (i.e., call by value versus call by reference). The fewer places in the code that can touch a variable, the easier it is to debug a program.

The competent programmer uses top-down design and bottom-up implementation:

> The first thing he does in any programming task is to analyze the entire problem into smaller problems that can be solved separately. He begins coding by writing functions that implement the primitives and building blocks he will need. The rest of the program almost writes itself. [D. Boundy]

o o o

We'll just reuse this data item.

The disciplined programmer does not give two names to one thing nor attribute two things to one name. Names are meaningful and specific, and their length is proportional to their scope. A loop variable used only once in a two-statement loop may be called "i", but a global variable that may be used anywhere in the program will have a long name that accurately describes its usage. [*Boundy's Laws of Naming*]

The disciplined programmer adheres to the standards of his workgroup, even when these standards appear arbitrary. A standard as small as indentation style makes the code more readable for a person who doesn't know it. That person might be the programmer, two years later, after the code is forgotten.

A program is written once, but read many times. Why not make it easy on the next person who reads the code?

o o o

That could never fail--don't bother testing for it.

When he first began to program, one of the authors of this paper (please don't ask which) developed an unfortunate impatience with writing code to test for error conditions. In his programs, he would skip error checking after certain operating system calls, such as Fclose and Fgetinfo which he "knew" could never fail.

Then he discovered that a :File Command could cause the Fclose Intrinsic to fail, leaving the file hanging open. And Fgetinfo failed once, undetected, when he closed the file by mistake, and another time when it was accessing a remote file on another system, and a third time when the calls were being intercepted by another vendor's run-time library.

This programmer learned the hard way to test every system call and every library function for failure. If he doesn't, his program may go merrily along, processing with the wrong data. He accepts the fact that most of his programs will have more lines of code to handle failure than to handle success.

Sometimes we develop a similar attitude toward manual procedures, such as installing a new version -- "I've done this so many times, I could do it in my sleep!" Implying, "I couldn't possibly make a mistake". Unfortunately, uninspiring tasks are the most likely place to misstep, since it is difficult to keep your concentration focused. When we install new software, we have to follow a written checklist. The checklist includes every step to create a new version of the product, plus steps to verify that the installation was done properly (i.e, make a demo tape and install it on another machine).

o o o

Fascinating project -- too bad it failed.

Fascination with grandiose schemes and bigness leads to **White Elephants**. The symptoms are pretentious objectives, high cost, fantastic claims, neglect of other projects, fanatical denial of failure, and a sudden, total write-off. All so unnecessary. In software, big results can come from tiny investments and tiny results can come from big investments.

> The original Lotus 1-2-3 (tm) and dBASE (tm) programs were two of the most successful application programs ever written. 1-2-3 was written mostly by one person in eighteen months. The macro capability, one of the things that made 1-2-3 really successful, was added by the developer at the end because he had some extra time -- it wasn't even in the informal spec he had. dBASE was written by one person over a two-year period while he also held a full-time job. [D. Thielen]

Hewlett-Packard has done two large, exotic database projects for the HP 3000 that were never released to users. Those projects consumed R&D resources that could have provided badly-needed enhancements to HP's popular but untrendy IMAGE product.

> There is a popular view that technology is only technology if it is high-tech -- sort of a "Big Bang" theory of technological development where somebody suddenly thinks of a major innovation or invention. But most of the technological change that goes on in society is not the "Big Bang" type. Rather it is small, gradual, marginal things. [Michael Bradfield, Dalhousie University, *Globe and Mail* newspaper.]

Robelle's biggest failure was the Virtual Fortran compiler. We had no Fortran expertise and no local test sites, but we did have big plans. We were going to run big, scientific Fortran programs on a 16-bit HP 3000, instead of waiting for a new RISC computer. Although we did complete the project, it was late and never performed fast enough. We were seduced by the glamour of the project. We squandered two man-years we could have spent on more humble but more successful projects.

o o o

We tested it once by hand, isn't that enough?

You can't test for the complete absence of bugs, because you can't try every path through the code. But you can test the typical cases and the boundary cases: minimum value, maximum value, and no value. In rigorous testing environments, such as software for jet aircraft, it is standard practice to measure the percentage of program statements being executed by the tests and aim for 100% coverage.

Automated testing is the answer. The computer can do more tests than we can manually and do them more reliably. We borrowed the idea for test jobs from the Pascal validation suite, a series of tests that told whether a Pascal compiler was up to the standard. A typical job tests one command, often by modifying data two different ways and comparing the results. For example, copy a file with Suprtool and again with Fcopy. Any difference and the job aborts.

When we are revising a program, we schedule the test suite to run at night. It is amazing how often a seemingly minor change causes 10 or 20 test jobs to fail.

When working on a bug, a good practice is to add a test that reproduces the problem. When the bug is fixed, the test passes. Reproducing a bug in the test suite also provides a warning if the bug creeps back into the code. It is embarrassing how often old bugs resurface.

o o o

It's fixed, but is waiting for the next release cycle.

Installing new software into production is an error-prone process. The program has to match the source code, the help file, and the manual; everything has to be tested; and so on. The more error-proof the installation process, the more onerous and time-consuming it becomes. This discourages frequent software updates, causing release cycles to stretch out until it takes two years to get the simplest bug fix into production. Look how many years it has taken HP to add support for IEEE floating-point numbers to TurboIMAGE.

The way to quicken development cycles is to **automate every step in sight.**

For our products such as Suprtool, we have a job stream that regenerates a new version. The job stream

- recompiles all the code, including supporting modules and programs,
- runs the test suite against the NM and CM versions of the product,
- reformats the documentation files to check for hyphenation errors, which it delivers to the programmer through electronic mail, and
- generates a new help file.

Now if we could figure out a way to have the computer actually fix the bugs and write the documentation, we could retire.

o o o

It can't be changed, too much code references it.

Global variables are the worst enemy of good software. This insight came to us in two waves.

First we learned not to use a literal constant, such as "80", when an identifier like "buffer-size" was allowed. Which would be easier if you want to change the buffer size?

Later we learned that this is not enough. The wider the access to any data structure, the more code to be checked when that data structure is revised.

For example, Suprtool has a fixed-size table for the Extract Command. On Classic machines, the space used by Extract limits the space left for buffering. We want to convert the Extract table into a linked list. So far we haven't found the time, because there are so many places in the code where the table is indexed as an array. If we were doing the Extract Command today, we would hide the data structure in a separate module. The rest of Suprtool would call procedures in that module to access the data structure.

The AIFs for MPE XL are a good example of making programs data independent. An AIF is a fast subroutine for accessing system tables. When a system tool uses AIFs, it doesn't know the location and structure of system tables. MPE XL may be improved and changed drastically, but the system tools will still run properly.

o o o

That would mean changing all the programs.

The year 2000 haunts those of us with only two digits reserved for the year instead of four. Will our invoices be dated January 1st 1900 at the turn of the century? How are we going to sort by date when "01" is less than "99"? Why did the millennium have to occur in our generation?

We wish now that we hadn't hard-coded the date format into all those programs, nor sprinkled ad-hoc code throughout our systems to edit and format dates. Couldn't we just extend the year 1999 instead of going to 2000?

With perfect hindsight, we would have used modular programming in the first place. Programs wouldn't "know" about dates--they would depend on a **date module** for that knowledge; a module that holds all the functions and data structures for dates and reduces them to a clean, published interface; a module that edits dates, converts them between different formats, compares dates, and does date arithmetic. To change a date format, we change the module.

There is still time to correct past follies. You can break up large modules into smaller ones, write new, generalized modules to replace the old, restrictive ones, and design modules to be used as tool kits upon which other can build.

That should be enough to redeem past sins.

o o o

We gave the users what they asked for.

As an experienced developer once quipped, "**The firmer the specs, the more likely to be wrong.**"

It is natural to plan for only one release of a new program. Unfortunately, the original software design is seldom what the users really need. The harder we pressure the users to tell us what they want, the more frustrated we both get. Users can't always tell us what they need, but they certainly recognize what they **don't** like when they see it.

The key to writing quality software is to **do it in several releases.** Once the users have the first release, the programmer incorporates their feedback (i.e., complaints) into the next release. At our firm, we send new "beta test" software to our customers every month. During a year, we may have 80 test installations for a product with 800 active users -- about 10% of the customer base.

At Robelle we use a development method called "Step by Step", created by Michel Kohon. It breaks a large project into small, two-week steps. This does not mean ignoring long-term goals, but once we learn more about the users' actual needs from each step, we commonly adjust our long-term goals as the program evolves.

> The final aim is the program, not the analysis. So, until the program or its results are in the hands of the user, nothing is completed. [M. Kohon]

o o o

Computing Your Score

That is the last of our warning signs. Did you recognize many of them? Now calculate your score? Remember, five points per warning sign, and the lower the score, the better.

0 is too good. You must have cheated.

20 is excellent. Congratulations.

40 is respectable. Good work.

60 is worrying.

80 or more is a disaster. Update your resume.

Suggested Readings

Boundy, David. "A Taxonomy of Programmers." Software Engineering Notes (ACM SIGSOFT), October 1991.

Denning, Peter J. "What Is Software Quality?", Communications of the ACM, January 1992, Volume 35, No. 1.

Feynman, Richard P. "Personal Observations on the Reliability of the Shuttle." **What Do <u>You</u> Care What Other People Think?**, New York: W. W. Norton, 1988.

Gomory, Ralph. "Of Ladders, Cycles, and Economic Growth." Scientific American, June 1990.

Green, Robert M. "Improving Software Quality." **Steps to Software Quality.** Robelle Consulting Ltd., 1990.

Halberstam, David. **The Reckoning.** New York: Avon, 1986.

Kernighan and Plauger. **Elements of Programming Style.** Bell Telephone Labs, 1974.

Kohon, Michel. "Introduction to Step by Step." **Steps to Software Quality.** Robelle Consulting Ltd., 1990.

Peters, Tom. **Thriving on Chaos.** New York: Harper and Row, 1987.

Reich, Robert. "The Quiet Path to Technological Preeminence." Scientific American, October 1989.

Thielen, David. "Unconventional Thoughts on Managing PC Software Development." Microsoft Systems Journal, May 1991.

Paper # 3102

An Update on Software Inspection Methods

Steven M. Cooper
Allegro Consultants, Inc.
2101 Woodside Road
Redwood City, CA 94062
U.S.A.

Phone: (415) 369-2303
FAX: (415) 369-2304
UUNET: scooper@allegro.com

ABSTRACT

Companies spend increasing percentages of their EDP budgets on software each year, just as their reliance on software becomes greater and greater. Finding defects in software at an early stage of its life cycle is typically a painless, inexpensive process. Finding the same defect after the software has gone into production can be devastating. Unfortunately, most companies do not do all they can to find defects early, paying the price of late discovery.

This paper examines recent advances in software inspections, a method that has evolved from older methods, such as code walkthroughs and structured walkthroughs. It highlights methods described by Gilb, Fagan, DeMarco, and Yourdon, that are in use in companies such as JPL, IBM, ICL, GE, and HP.

THE PAPER

The Problem

Before someone can legally cut your hair or install a new light switch for you, they must obtain a license, having had some degree of training and proven some level of competency. Yet, very few companies have gone broke because their president got a bad haircut or some wacky electrician wired a light switch wrong. On the other hand, to become a programmer, one need merely announce to the world that they are a programmer; no license, no certification, certainly no proven level of competency. What risk does this programmer pose to our company with the president with the bad haircut and the light switch that turns on when it should turn off?

Things would be bad enough if all we had to worry about were our junior programmers. But in fact, some of the most disastrous bugs have been keyed by some of the most experienced programmers around. In the past few years, we've seen massive parts of our country unable to make long distance phone calls, due to an accidental, one-line bug in software. We've also seen huge networks of computers brought to their knees due to an intentional virus exploiting an unintentional hole in software. Surely, every reader of this paper can add other examples of system failures, program aborts, and just plain bugs that have plagued their own systems.

These are not new problems. Although Grace Hopper's moth, the original *bug* found in the world's first computer, was tangible, most of us are far more familiar with the intangible kind. Back in those days, building hardware was an art and testing was primitive. Today, building hardware has become a science and testing extremely sophisticated. Hardware "bugs" have fortunately become a pretty rare event. But software development stubbornly remains an art. And as software systems become more and more complex, these hidden bugs become harder and harder to find and offer higher and higher degrees of risk to their users.

History and Evolution

Throughout the years, companies have tried various methods of peer review in an attempt to find bugs before they go out the door. Many of you are probably familiar with so-called code walkthroughs. Here, the programmer circulates listings of his/her code a few days before a review. Each reviewer looks over the code and then they all meet to collectively beat up on the poor programmer. Jokes aside, this method has uncovered many bugs through the years and can be quite effective. Unfortunately, all too often it degrades into a *let's all attack the programmer* session or a lengthy debate on some triviality, such as should you indent three or four spaces after a BEGIN. There is also the problem that as deadlines approach and pressures increase, walkthroughs are often the first thing to get cancelled. Of course, this is probably the time that they are most needed.

In 1976, Michael Fagan[1] wrote of a new type of review being used at IBM. Often referred to as Fagan-style inspections, this method is characterized by the presence of a *reader*, a person whose job it is to paraphrase the document being reviewed at the inspection meeting. These inspections were finding new classes of bugs and at the time were found to be the most cost-effective method of finding defects.

Note the use of the word *document* in Fagan-style inspections. No longer is this just a method of reviewing code. Fagan proposed that anything that could be printed on paper could and should be inspected. This includes such things as investigation reports, program specifications, blueprints, user manuals, test plans, installation instructions, and engineering drawings, as well as source code. In fact, studies will show that source code is the least desirable thing to inspect. The sooner in the life cycle of a product that a defect is discovered, the cheaper it is to fix. Hence, investigation reports, design documents, and internal specifications should all be inspected before any code is written. When this is done, inspecting the code becomes much simpler: does the code do what the specification said it should. Uncovering a major design flaw after the code is written is expensive indeed. One reason that it is so expensive to find defects during the final system test phase is not the time it takes to fix the bug, but instead, the time it takes to rerun all of the system tests again and fix new problems that the fix may have caused.

Despite all of its positive aspects, Fagan-style inspections still had their drawbacks. For one, it is very easy for someone to come unprepared to an inspection meeting and coast, letting the reader do all of the work. Without spending in-depth preparation time, more complex, deeper defects will often go undetected.

Much later, in 1985, another IBM employee, Carole L. Jones[2] , wrote of a different technique in use to improve quality by preventing defects from occurring in the first place. This technique relies on causal analysis to determine the cause and feedback to prevent its recurrence.

Then, in 1988, Tom Gilb[3] presented an enhanced inspections method that combined the best qualities of the Fagan and Jones methods along with improvements discovered while working with his consultation clients, including such companies as McDonnell Douglas, IBM, and ICL. This inspections method has continued to evolve. It is this latest version that is in use at Hewlett-Packard, is taught by Allegro Consultants, and is described below:

The Inspection Process

We define an inspection as follows: a formal review of a document by the document owner and a team of peers looking for errors, omissions, inconsistencies, and areas of confusion in the document. The word *formal* is important; this process has seven well-defined steps and rules that must be followed. Note that the *owner* is involved. The Owner is the person responsible for the quality of the document and is the person that will do the rework. This is often the document's author, but in the real world of reorganizations and promotions, this is not always the case. We will also borrow the definition of *document* mentioned above.

As for the *team of peers*, each member of the inspection team will take on one or more of the following responsibilities: One will serve as Moderator, taking on overall responsibility for this inspection through all seven steps. One (or more, but one at a time) will serve as Scribe, logging the required information in the Logging Meeting and Cause/Prevention Meeting. Most (or all) will serve as Inspectors, searching for the *errors, omissions, inconsistencies, and areas of confusion*.

The process itself is divided into seven steps, as follows:

Step 0: Planning
In this step, the Owner and the Moderator start by working together to ensure that the document is ready for inspection. This is done by checking the Entry Criteria, a set of conditions that must be met before a document can be inspected. They then choose a team, prepare the Inspection Packets containing the document to be inspected and perhaps some related documents, and schedule the remaining six steps. They also identify special objectives for this particular inspection; in other words, what we are hoping to accomplish with this exercise.

Step 1: Kickoff
This is the first step in which the inspectors get involved. This is typically a short meeting in which the Inspection Packets are distributed and explained, the objectives for the inspection are described, and the team commits to the inspection schedule.

Step 2: Preparation
In this step, each inspector works on their own, inspecting the document, looking for errors, omissions, inconsistencies, and areas of confusion. Typically, each inspector is asked to spend one to two hours on this step, depending on the size and complexity of the document being inspected.

Step 3: Logging Meeting
This is the meeting in which the items found during Step 2 are logged by the Scribe. This log is called an Item Log. This meeting is facilitated by the Moderator, who keeps the meeting focused on problems, not solutions. Discussions are allowed only as long as the Owner requires to understand the item being logged. This meeting typically lasts up to one and one half hours.

Step 4: Cause/Prevention Meeting
This meeting immediately follows the Logging Meeting. The Logging Meeting is aimed at improving the quality of the document being inspected, whereas the purpose of this meeting is to improve the quality of future documents. In brainstorming fashion, one or two items are chosen for discussion and possible causes are suggested. For each potential cause, possible solutions are suggested that would prevent this type of item from

occurring in future documents. This meeting is typically limited to one half hour.

Step 5: Rework
This is the step in which the Owner takes the Item Log from Step 3 and addresses each item. *Addressing* might mean to fix the item, raise an issue to management, or decide not to make a change. The Owner has responsibility for the quality of the document, so the Owner has ultimate say as to what is and what isn't a defect.

Step 6: Follow-up
The Moderator and the Owner verify that all items have been addressed and all statistics on the inspection have been collected and reported. Then, the Exit Criteria are checked. Exit Criteria are a set of measurable items, previously agreed to, that are used to determine if the document passed the inspection or not. If they are all met, the document is marked *EXITED*. If not, it is marked *NOT EXITED*. In this latter case, the document may or may not be reinspected. If it is to be reinspected, a new inspection will begin again at Step 0.

The process also contains a number of other special features to ensure that a maximum number of major defects are uncovered in minimal time. For instance, the suggested size of a document to be inspected is 1 to 15 pages, with an ideal size of 4 pages or, in the case of code, 150 to 200 lines of code. Documents that are larger than this are *chunked* into portions to be inspected in separate inspections. This may seem like a small size, but a lot of research and experimentation has gone into determining that this is the best size.

Another special feature is that of the code of silence. What goes on in an inspection is private and is not discussed outside of the inspection, with only a few exceptions. Management receives summary information on the inspection program: hours spent, total defects found, documents inspected, documents exited, etc. They are not given statistics on individual inspections, nor are they allowed to attend logging meeting, unless they are acting as inspectors, have done the preparation, and are willing to live by the rules of the process. For the inspection process to work, it can not be contaminated by mixing in other objectives such as employee evaluation.

Conclusion

Software review techniques have improved significantly in the past twenty years. Companies are finding that fully implementing an inspections program produces a higher quality product that is easier to maintain and support, without adding to the development time schedule. Properly executed, an inspections program can cut testing time in half. Typically, time to find and fix defects uncovered by inspections is 10 times shorter than for black box testing. And, it has been shown that an inspections program can find 75% to 90% of the defects prior to testing. All of this will lead to a 14% to 25% increase in productivity[4].

In addition to the primary goal of producing higher quality documents, other benefits are also realized. A joint sense of ownership for the product as a whole often develops. Also, there is a high degree of training going on, as more junior people see best practice examples from more senior people and everyone sees what is important to the others. With the cost of software development and maintenance being what it is today, and with the demands for quality that the world has come to expect, implementing a formal and rigorous inspections program has become a necessity for any company that wishes to remain competitive.

Footnotes and References

[1]Fagan, Michael E.: "Design and Code Inspections to Reduce Errors in Program Development", IBM System Journal, Vol. 15, No. 3, 1976.

[2]Jones, Carole L.: "A Process-Integrated Approach to Defect Prevention", IBM System Journal, Vol. 24, No. 2, 1985.

[3]Gilb, Tom: "Principles of Software Engineering Management", Addison-Wesley, Reading, MA, 1988.

[4]"Findings on Inspection Techniques", 14th Annual Nasa-Goddard Software Engineering Workshop, Nov. 1990.

Paper #3103

STREAMX: It's Not Just For Passwords Anymore!

Stephen S. Hammond
Systems Analyst
Association of American Medical Colleges
2450 N Street, NW
Washington, DC 20037 USA
☎ *(202) 828-0448*

One of the features bally-hooed with the arrival of MPE XL 3.0 was the ability to stream jobs which do not have embedded passwords. The job is streamed and the user streaming the job is prompted for the password of the user in the job card.

Gee, we've been fending off DP auditors about this for years and now we have it!! I teach tutorials on VESOFT software and had a student tell me he had a DP auditor give him 'demerits' for having embedded passwords in his job streams on his HP3000, but not penalize him for having embedded passwords on the IBM mainframe. The reason was because the auditor knew "how hard it is to change passwords on the IBM".

Somewhere someone is missing the point! If an embedded password is a security risk on one computer, it must be a security risk on all computers. But that's another issue. I'm here to talk about how you can use STREAMX to do a lot more than just prompt you for passwords. If you use none of the features of STREAMX other than replacing passwords, it will be better than the MPE XL method of eliminating embedded passwords, but why use only part of what you have? Do you buy a television and only watch one channel? Do you only use Windows™ to play solitaire?

I'm going to tell you how to use STREAMX to do more than just get rid of the passwords. I like to call them my 'Top Ten Reasons for Using STREAMX':

☞ 10 - You won't get those pesky password prompts

STREAMX is intelligent. It checks the capabilities of the streaming user and if that user can determine the password through normal security procedures, then why prompt for the password? MPE XL will prompt you for passwords no matter who is streaming the job, which brings us to...

☞ 9 - Do you really want your operator to know the MANAGER.SYS password?

The answer is almost always no. MPE XL's new streaming utility prompts for passwords whenever none is present in the job card. STREAMX has a very handy control file called STREAMX.DATA.SECURITY or maybe STREAMX.DATA.VESOFT if you're running version 2.4 of the VESOFT products. *(Author's Note: From this point on, I will refer to this control file as STREAMX.DATA.)* In this file you can declare users who can stream a group or groups of jobs without being prompted for passwords, and the most likely choice in this category is OPERATOR.SYS. The keyword $NOPASS in this control file sets up the user, the fileset he can stream and the logon id of the jobs he can stream.

```
$NOPASS   OPERATOR.SYS   @.JOBS.PROD   @.@
$NOPASS   SOURCE.DEV     @.JOBS.DEV    COMPILE.DEV
$NOPASS   DAVID,@.@      @.@.DEV       @.DEV
```

These basically say that OPERATOR.SYS can stream any job in JOBS.PROD, no matter how those jobs log on. The user SOURCE.DEV can stream any file in JOBS.DEV which has a logon id of COMPILE.DEV and any session with the session id of DAVID can stream any file in the DEV account which has DEV as the logon account in the job card.

☞ 8 - Unlock the lockwords

STREAMX lets you eliminate lockwords in jobs by either prompting the streaming user for the lockword at stream time, or with the use of the **$NOPASS** keyword followed by a **$NOLOCK** keyword in an entry in STREAMX.DATA to eliminate those prompts.

!JOB QSRUN,....
!RUN ?$LOCKWORD=QUERY.PUB.SYS$?
will prompt the user:
What is the lockword of the file 'QUERY.PUB.SYS'?

or
$NOPASS OPERATOR.SYS @.JOBS.@ @.@ $NOLOCK @.PUB.SYS

will eliminate any lockword prompts on all lockworded files in PUB.SYS for OPERATOR.SYS streaming jobs in any JOBS group on the system.

☞ 7 - Nested jobs won't fly the coop

In MPE XL, any job which is streamed by an earlier job must have embedded passwords. The operating system only looks at the first job and prompts for the password. Any subsequent jobs in the sequence must have embedded passwords to run.

STREAMX checks all the jobs in a sequence at the time the first job is streamed. The user will be prompted for any and all required passwords at that time. It is even possible to force STREAMX to ignore errors in subsequent jobs in a nest if necessary, such as the first job creates an account which is the logon account of the second job, or better yet, the first job builds the file which is the second job. To do this:

:SETJCW STREAMXPERMNONESTERROR=1
to set the JCW permanently, or

:SETJCW STREAMXTEMPNONESTERROR=1
for only one job.

These JCWs tell STREAMX to ignore any errors in the subsequent jobs, such as non-existent account, non-existent file,etc.

You can also use the JCWs STREAMXPERMNONEST and STREAMXTEMPNONEST, which inhibit STREAMX from looking at any job in the sequence after the first. This is helpful when the first job modifies the file which is the second job.

In both circumstances, STREAMX will stream the second job, so you need to ensure that either the logon user in the first job has the capability to stream the second job without a password prompt (log on with SM capability) or explicitly name the files in a $NOPASS entry in STREAMX.DATA.

☞ 6 - Avoid the 'big truck' syndrome

Does your shop have one of those people who knows how every job is supposed to run but has never taken the time to write any of it down? What happens if that person gets run over by a big truck? You're in the middle of end-of-year closing and none of those jobs will run (can you say 'Chapter 11'?). Have 'Mr. Everything' modify all those jobs to use STREAMX to make the jobs 'smart'. STREAMX has a powerful expression environment which makes jobs 'smarter'. Do you have several jobs which run the same program with different input options? Use STREAMX expressions in the job and have it prompt the user for the proper options.

Statements preceded by a '::' or variables within '{}' are resolved prior to submission of the job.

```
!job STATERPT,MANAGER.ACCTREC
::prompt string STATE="What state is this run for"; &
::       default="ALL"
!file....
!RUN STATPROG
{STATE}
!eoj
```

In this job, a program processes data for a particular state or all the states. In the MPE XL environment, the standard way to handle this would be by changing the line after the '!RUN' statement in the job stream every time the job needs to be run. As shown here using STREAMX, you can prompt the user for the state at the time of submission and if the user just gives a carriage return, the parameter 'ALL' is used.

At my association, we often print membership directories using a COBOL program to produce the file and LPS to do the actual printing. Directories have a habit of having introductory pages which vary in number from year to year, so the user would tell us what page to start the directory on and then someone (usually me) would change the job stream. Instead, I let the user stream the job using STREAMX and prompted the user for the page number and the number of copies. And since the user usually runs this job several times to produce 'proof' copies of the directory before the final run, I put a default starting page number of 1. Presto! The user is now responsible for all the input parameters.

```
!job CASDIRCT,casuser.diracct
!comment...
!file...
::prompt integer PAGE = "What page do you want to begin on";&
::              default=001
::prompt integer NUM = "Number of copies of the directory";&
::              default=1
!run CASDIRCT.OBJECT
{strwrite(PAGE:3:'zerofill')}
!file LPSOUT=CASDRCT;dev=laser,8,{NUM}
!run LPS.PUB.SYS
2680A
CASENV.OBJECT
N
CASDRCT
!eoj
```

This job prompts for a starting page number (PAGE) and a number of copies (NUM). The line after the '!run' command writes the numeric variable PAGE as a string, making it three places long and filling it with leading zeroes. It then prompts for the number of copies to be printed by LPS and places that number in the file equation.

☞ <u>5 - Nothing is foolproof because fools are so damn clever!</u>

So you've made this nice 'smart' job and then you turn it over to some stupid user to run. You need to be able to check the input and reprompt or even exit without streaming the job. Again STREAMX gives you that power.

```
::prompt integer PCT="What sales percentage are we using";&
::      check=(BETWEEN(PCT,4,9);&
::      checkerror="Sales percentage must be between 4% and 9%";&
::      default=5
!job monthly,....
!run ...
{PCT}
!eoj
```

In this case the monthly sales report uses a variable percentage which changes every month. This job prompts the user for that percent, but still establishes an upper and lower limit (inclusive). That way some wahoo in sales can't run the monthly report to impress some bar buddy with inflated figures. If that range changes, a modification to the job stream is simple.

```
::prompt string INP = "What data file do you want to use"
::if not FEXISTS(INP) then
::   echo The input file {INP} does not exist!!
::   echo The job has not been submitted!!
::      exit
::endif
!job...
!file input={INP}
!run...
```

Here the user is prompted one time for the name of the data file and then the VESOFT file function FEXISTS is used to determine if the file exists. If the file is not there, the job will tell the user the file doesn't exist and terminate before it is ever streamed. If the file is there, the file name is inserted into the file equation in the job.

☞ <u>4 - Loop the loop!</u>

I just showed you an example of the IF...ELSE...ENDIF processing, but you can even do WHILE...DO...ENDWHILE processing to loop through a prompt any number of times.

```
!job...
!run QUIZ.CURRENT.COGNOS
use SCHOOLUSE
::prompt string SCHLNUM="Enter school code or // to exit"
::while SCHLNUM <> "//" do
{SCHLNUM}
::      prompt string SCHLNUM="Enter school code or // to exit"
::endwhile
::assign FILL='   '
{FILL}
!eoj
```

In this case the QUIZ 'use' file has a CHOOSE statement which will keep prompting as long as there is any input other than ' '. The reason there are two

"::prompt" commands is this is just like a programmatic primary and secondary read. There must be something in the variable which the 'WHILE' tests when it starts. Then the STREAMX ASSIGN command is used to force a blank line to terminate CHOOSE in the QUIZ code. This way the job can handle one school number or twenty-one.

☞ 3 - MPEX is lurking in the background!

With the release of version 2.3 of the VESOFT software, you are able to do MPEX commands from a STREAMX job without running MPEX.

```
::if vefinfo("MCATDATA.PUB.SSSAMCAS").opened
::      echo The MCATDATA file is in use!!
::      echo The job cannot be run!!
::      echo The following jobs/sessions have it open.
::%     listf MCATDATA.PUB.SSSAMCAS,access
::      echo Contact them and then resubmit the job!
::      exit
::endif
!job MCATLOAD....
```

In this case, I used the VESOFT file function VEFINFO("filename").OPENED to determine if a very large and very important file is open. If the file is open, the job tells the user, via ECHO commands, that the file is open, then does an MPEX LISTF,ACCESS command to show who is using the file, and then finally terminates the job before it is streamed.

☞ 2 - Looks can be deceiving

My first career was as a sports writer. I spent about five years watching coaches and managers do their darndest to deceive the opposition. (I spent one summer covering George Allen, the master of deception.) If you view system security from the perspective of 'us against them', maybe a little deception can make a difference. If you just got STREAMX and you want to implement it right away, just change your passwords and start using it. STREAMX considers a bad password the same as no password. Leave those old passwords in the job streams. If someone is snooping for passwords, all they'll find are bogus passwords. On top of that you can use some keyword commands in STREAMX.DATA to restrict job submissions for certain accounts.

$LOGON-FORBID @.GL-MGR.GL @.JOBS.GL MONTHLY.GL

This prohibits anyone except MGR from streaming jobs in the JOBS.GL group which logon as MONTHLY.GL. Again we're just trying to prevent some of those untrustworthy employees from playing with the general ledger files.

☞ <u>1 - Avoid the OOPS</u>

STREAMX helps you avoid some of those embarrassing mistakes that crop up in our jobs. On more than one occasion, I've spent many hours testing a process in online mode. When I'm happy with the test, I set up the job stream and fire it off. But it doesn't work. I spend about the next five minutes to three hours looking at the STDLIST, having other people look at the STDLIST, trying to think who to call and then the bulb blinks on - file equations!! The STREAMX command **::SAVEFILEEQ** will pick up the file equations from your session and carry them over to the job.

Another nice feature keeps those sneaky programmers from running their compiles in the 'C' queue.

$JOBPARMS-ALWAYS @.DEV @.@.DEV @.DEV "PRI=ES"

This says that any streaming user in DEV, streaming any job in DEV, which logs on as any user in DEV will always have the job run in the ES queue, regardless of what the job card says.

$JOBPARMS-DEFAULT @.@ @.@.@ @.@ "OUTCLASS=LJ,9,1"

This says that any job which does not have an OUTCLASS parameter will be printed on the laserjet, with a priority of nine and one copy.

You can use PARMs when you run STREAMX to debug the job if it gets complicated (or you try to get too fancy). **PARM=%40** saves the stream to a file designated as STRMSAVE. Add %100 to the value (**PARM=%140**) and the file will be saved without embedded passwords, add %200, the file will be saved without lockwords. Thus "**RUN STREAMX.PUB.SECURITY;PARM=%340**" will save the file without embedded passwords or lockwords. Of course, on top of this, you can add %1 and make it a situation where STREAMX runs immediately and looks for the file designated as STRMFILE. But let's not get too fancy!!

☞ 1´ - And 11th in a list of 10

You bought it, use it! STREAMX comes when you purchase SECURITY/3000. You didn't have to pay extra, so it's a bonus. Use it's power and make your job streams smarter and your job a little easier.

Acknowledgements

Several individuals helped with the preparation of this paper and I would like to thank them: from the tech support staff of VESOFT, **Gil Milbauer** and **Paul Taffel** and, of course, **Eugene Volokh** and **Vladimir Volokh**; and from SMP Associates, **Sam Patsy**. I am also grateful to my supervisors, **Brendan Cassidy**, **Sandra Lehman** and **Maryn Goodson**, who have allowed me to learn and teach what I've learned. I also wish to acknowledge, again, and thank **Vladimir Volokh**, who has given me an opportunity to work with the people at VESOFT and to help others learn about their products.

User Interface Prototyping — for free!

Anthony C. Furnivall
SDL/Software, Inc
P.O. Box 601
BUFFALO NY 14213

BACKGROUND TO THE PROJECT

The research described in this paper was prompted by two things. First was a paper presented by Lisa Burns, at the Orlando INTEREX conference, which I will discuss in a minute, and second was a need to provide a rapid prototyping facility so that user interactions with a system could be explored. As the work developed, it became clear that the results were of sufficient interest to warrant sharing them at a wider level.

Previous paper by Lisa Burns

Lisa Burns' paper in Orlando described the use of freely available resources to prototype a user-interface. The paper was exceedingly well received, and it provided a stimulus for my own activities. In the paper Lisa showed how, by using the ENTRY program, a set of user interactions could be modelled, including the display of data values on the screen, and the progression from one screen to the next. Because no extra programming was necessary, this meant that a prototype could be put together in a relatively short period of time, and shown to users for their comments.

The idea of bringing together these freely available resources was a new one for me, and I found it very exciting. Indeed, I used it on a couple of projects, and was able to come to a faster agreement with my clients about how the user interface should work.

Benefits

Lisa's approach has two big benefits - the ability to prototype the user interface, and the use of freely available resources.

The effort to build a complete application in order to test the user interface is prohibitive. I had previously developed a program to examine VPlus forms in their natural condition (rather than the artificial environment of FORMSPEC), but this program had limited capability of modelling an entire user interface – Lisa's approach allowed a complete forms sequence to be shown. Effectively what she did was to script the user's actions, and to model the response from the program.

This approach leveraged the presence of free software. VPlus, FORMSPEC and ENTRY are all provided as part of the Fundamental Operating Software. This meant that no extra expense would be incurred for acquiring the prototyping capability. In a world of tight budgets, this is an important consideration.

Drawbacks

However, there were also some drawbacks to the approach described in Lisa's paper. Chief among these were the amount of time taken to develop the prototype, and the relative inflexibility of the approach.

It takes a lot of time to develop a form, and when this form must be copied several times to allow for the modelling of various user actions, the time involved can easily become significant. Unfortunately, the investment in developing the extra forms can not be recovered, because the additional copies will not be used in the application. Ideally I wanted to find a way to use as much of the prototype as possible in the final application.

The approach described by Lisa was based on the use of the ENTER key to move from one form to the next. This is because the ENTRY program reserves the meaning of function keys to itself. Many applications make use of the function keys, and if the pressing of a function key can not be prototyped, then the value of the prototype is diminished.

VPlus user action	Supported by ENTRY
ENTER key	Yes
f1...f8	Reserved for own use

Table 1 - VPlus user actions, and how ENTRY uses them

The fact that ENTRY reserves these actions is indicative of the fact that they are useful actions! If this is so we need to be able to provide them to our users, and yet the approach of using ENTRY specifically excludes them. The new program should certainly allow for the use of function keys as part of the user interface.

New approach

In coming up with a new approach, then, it seemed desirable to build on the positive aspects of Lisa's ideas, and to see what could be done to mitigate the draw backs.

To extend the ability of the prototype to react to different responses, I had to find a way of controlling the response of the modelling program. This meant finding a way of reading a file of variable length data items, and providing an easy way of maintaining this file.

The Fundamental Operating Software contains a unique resource, in the form of Native Language catalogs. These catalogs are actually a variable record length database management system. Each 'record' in the database has a unique key (the SET and MESSAGE numbers), may be of variable length,and can be retrieved directly. In addition, there is a freely available means of maintaining the database, in the form of the GENCAT program.

Perhaps there was a way to use the catalogs to drive the prototyping process...

A replacement for ENTRY was a trickier proposition. However, the requirements for such a program were fairly simple, and it soon became an obvious decision to build this program from scratch. The only problem was how to make it freely available, and the equally obvious decision was simply to do just that - to make it available at no charge, to anyone who was interested!

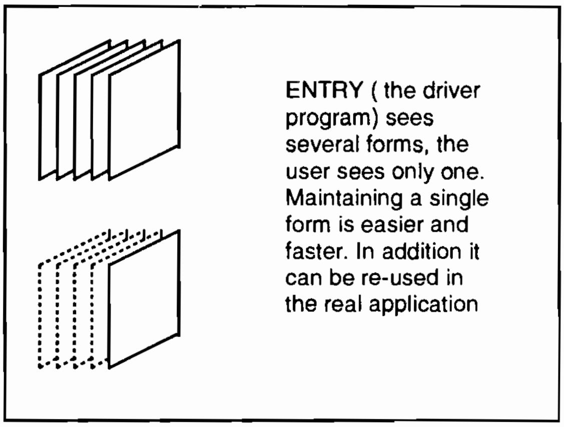
For details on how to get a copy of this driver program, see the notes at the end of the article

Mitigate the drawbacks

So much for the easy part. Now I had to deal with the two problems with the earlier approach - making the change process faster, and providing greater flexibility in the user's responses.

Providing a faster change capability

The bulk of the effort in the approach described by Lisa comes in the need to make additional copies of the forms, and to include extensive initialization values. In addition, these extra copies are not generally usable in the final product. While FORMSPEC is not the fastest tool in the world, it does represent a minimum investment in creating a form. My decision was simply to create the one version of a form that would be used in the final application, and to have the driver take care of everything else.



ENTRY (the driver program) sees several forms, the user sees only one. Maintaining a single form is easier and faster. In addition it can be re-used in the real application

Fig. 1 - Reducing the number of copies of each form

This meant that 100% of the investment in prototyping could be recovered in the final system, a desirable outcome. In addition, it meant that changes in the sequence of user actions would not require that forms be created, or removed. The changes could take place in the scripting resource. It is much easier to change a message catalog than to change a VPlus forms file!

Providing greater flexibility in modelling the user interface

By giving up the use of ENTRY, it became possible to allow the user complete access to the various parts of a VPlus display. This meant, most significantly, allowing function key presses to have the same visible effect that they would have in the final application. By doing this, I could model all the actions that a user might take.

A significant difference between this approach, and the one that Lisa described back in Orlando, is that the data values used by the program are not visible. For applications where data values drive the program, rather than simple button presses, this may be a problem. An example of such a situation could be the use of a special mode or command field as part of the form, rather than as a function key label.

The techniques described here are *not* useful if your application uses a data-value driven approach.

Success criteria

After developing the VDEMO prototyping system, it was important to review the success of the research that I had carried out. The best measure of my success was that not only have I successfully used the methodology to prototype the user interface for a large vertical market system, but have also been able to incorporate all of the resources used in the prototype into the resulting application code. There is no waste from the process!

THEORETICAL CONSIDERATIONS

Before going on to describe the steps involved in specifying and building the prototyping system, it will be necessary to digress for a short while, and examine the idea of using resources, the concept of modality in a user interface, and event-driven programming.

Using resources

A resource is, more or less, a piece of data which is *used* as a single unit by a program, to carry out its work, rather than a piece of data *operated on* by the program as part of its mission. The distinction is a fine distinction, but if we think of an external subroutine, or sub-program as a code resource, then the idea of using a form or a catalog entry as a data resource may be easier to grasp. The important thing to retain is that the user does not have to worry about the internal nature of the resource, all the details are taken care of automatically, by the resource management part of the program. Resources can be changed independently of the program, and the program then uses the new resources in the same way that it used the old ones.

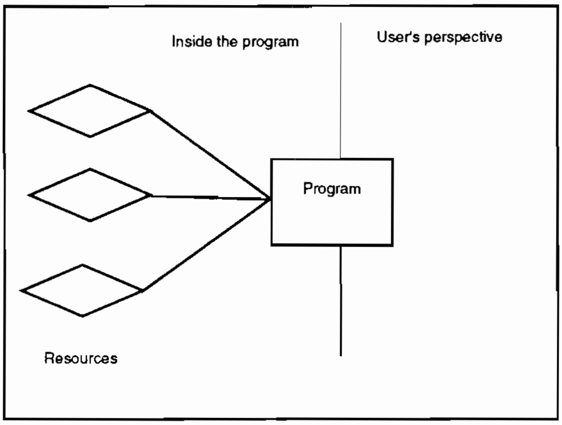


Fig. 2 - Use of resources by a program

User interfaces, especially the exotic ones I mentioned earlier, such as Windows, X-Windows, OSF/Motif and the Mac interface, are all great users of resources. Every visual element on a display is a resource, and the ability to control and change that element is improved if the resources can be managed separately from the main program.

If we are going to provide a means of prototyping the user-interface, then it is in our own best interests to make the resources of that user-interface as easily managed as possible. The method I am describing uses two sorts of resources - VPlus forms, and Native Language Catalogs (actually it uses the messages in those catalogs).

The critical idea about a resource is that is managed separately to, and externally from the program which uses it. All the program must do is access or use the resource in a well defined way. The actual nature of the resource is secondary to the program.

Modality in the user interface

Fashions in computing are less volatile than fashions in clothing, but they do change over time. A current vogue is to avoid a modal user interface. This term is bandied about quite freely, and may cause some confusion, so I would like to explain what I understand by the term.

When examining modality in a user interface, it will help to look at three ways of interacting with the user - the modal interface, the semi-modal (command line interface) and the modeless interface.

Modal interface

A modal interface occurs when the user is prompted to supply a value for a given data item. A typical exchange might be:

What is the value for π? **3.1415926535**

Here the computer has requested a specific value, and has waited for the user to respond. Until a response is received, the computer will do nothing. Actually, in a multi-programming environment such as MPE or HP-UX, the computer will eagerly seize the opportunity to go off and service other users, but from the point of view of the user in question, the statement is true.

The problem with a user interface which is modal, is that the user relinquishes control. This is considered a bad idea. The only way out of the dilemma is to provide several 'back-door' responses to the prompt. *E.g*:

What is the value for π? **exit**

This is certainly not a valid value for π, but allows the user's intent to be realized.

There is, perhaps, nothing intrinsically wrong with modal interfaces – indeed, there are times when such an interface is not only desirable, but necessary, in order for the computer to proceed. However such situations are likely to be in the minority, and it is not a good idea to model a whole user interface in such a modal manner, because control over the environment is transferred from the user to the computer. A good example of a modal interface which is desirable, is the dialog used in various diagnostic systems. However, even here the responses include the possibility of 'escaping' from the modal nature of the exchanges.

Semi-modal interface

A slightly less restrictive interface is the command-line prompt. This represents a semi-modal interface. After all, any of the valid commands (or indeed, any of the invalid commands!) could be typed in response to the prompt, so a greater degree of flexibility is gained.

However, the means available for the user to continue are limited to commands. Shortcuts such as loading up function keys to transmit different strings do not represent a change away from this type of interface. We are still limited to a single channel of communication, or a single style of communication.

Control here is passed back to the user, but only to the extent that the user can remember the current subset of permissible responses. An example of the confusion which can arise in a semi-modal interface is the inconsistency between the various ways of exiting a program which uses it. Do we type 'End', or 'Exit', or simply 'E'? What about 'Quit', or 'Q'? Our flexibility of response is retained, but we receive very little stimulus, or feed-back about what responses are available.

Modeless interface

A modeless interface provides no prompt, and permits a wider variety of user actions. This includes such actions as typing on the keyboard, pressing function keys, clicking with a mouse or other pointing device, and in general, including the ability to respond to any event in the external world.

This mimics much more the way in which people really work. For example, as I was writing this passage, I answered two telephone calls, and replied to a tape request on my main computer. The following table summarizes the three types of interface, and helps differentiate them from each other:

Interface	Prompt	Response
Modal	Tightly focussed, full of meaning	Single value, no other value
Semi-modal	Loosely focussed, limited meaning	Defined set of commands
Modeless	no prompt, complete state displayed	Open set of actions

Table 2 - Different styles in user interfaces

The modal interface, you remember, supplies a single tightly focussed prompt, of varying comprehensibility, to which a single reply (or a limited subset of replies) is permitted. The prompt completely defines the meaning for the response.

A semi-modal interface provides a much less focussed prompt (a colon ':' for example) to which a wider variety of responses can be made. These responses, however, are limited to a single channel - the command-input channel. The meaning of the response is contained within the response, and the program must interpret the response to extract the meaning. (This is why such programs are generally called Command Interpreters!)

A modeless interface provides no prompt, other than the current state of the program. It provides text entry areas, buttons, scroll-bars and menu areas to allow the user to control all aspects of the program. Such interfaces include Windows 3.0, X-Windows, and the Macintosh.

These interfaces all look very different from a typical 'glass teletype' terminal screen. This means that different methods must be developed to manage the actions of the user. The method that is used is called event-driven programming.

Event driven programming

Event driven programming arises naturally as a consequence of the modeless interface, and the need to respond to any action that the user cares to make. This differs dramatically from the typical procedural approach used in many commercial applications.

Procedural approach

In a procedural approach, the system designer specifies in minute detail the actions that the program takes at every turn. As far as possible, the data needed by the program is gathered at a single point, and is then processed by a hierarchical set of procedures. This leads to great efficiency when processing large quantities of data.

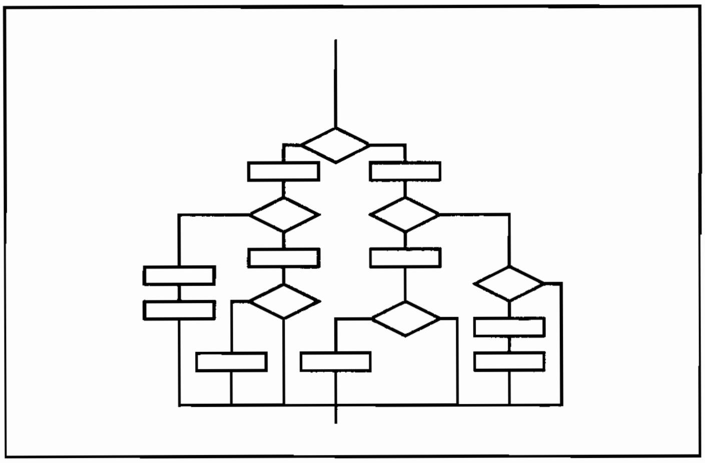


Fig. 3. Procedural design involves detailed planning

When reading a file to get a data record, for example, there is no need to post a prompt other than the FREAD request, or to worry about responses other than a successful return, or an End of File condition. Any abnormalities can be handled fairly easily, and the bulk of the designer's creativity can go into designing the work done by the application program.

Event-driven approach

With an event driven approach, we must be able to respond to any action the user is capable of making, from the helpful to the mildly perverse. In addition, we must at least give the appearance of being always ready to respond to whatever the user wishes. This means that rather than designing in large units, we must make each response as short as possible, so that we can get back and handle another action.

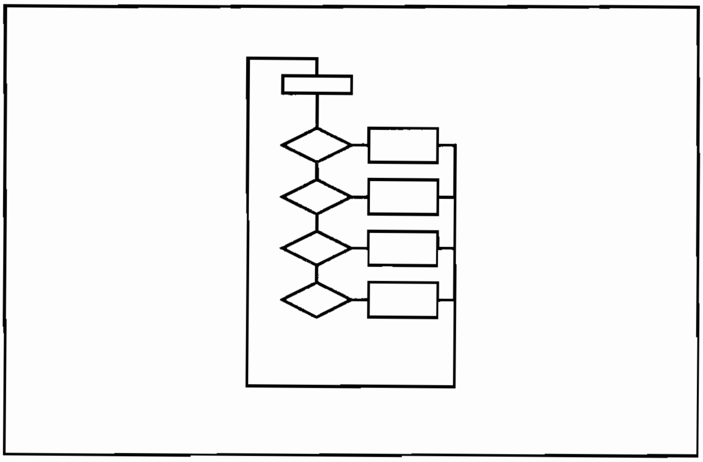


Fig. 4 Event driven design emphasises short event management procedures

This leads to the idea of a dispatching loop, where the next event is gathered, analyzed, and dispatched to the appropriate procedure to take care of whatever the user has requested. An added advantage of this is that it leads to modular code, since each of the responses can be made independent of the others. It also allows us to create events from within a response procedure. For example, if when I press a particular function key, I want to have the program move to a different form, and then behave as if the user had pressed a second key, I can push the desired event into the queue, and have the program process it as if the user had in fact typed it.

FUNCTIONAL REQUIREMENTS

In order to determine the functional requirements for the prototyping program (which I shall call VDEMO after this), we need to examine the various pieces of screen 'real-estate' and see how they can be used. In an GUI environment, the screen is a blank sheet onto which we can place a wide variety of different controls, and other widgets. With VPlus, our chosen interface technique, the choice is rather more limited.

What areas of a screen are available for use?

VPlus allows us to get at 3 different areas of the screen. In addition, because we can gain access to the terminal file, through the VPlus Comarea, we can also gain access to the terminal function key area, for the purpose of writing a simple message. This message is cleared by simply hitting the Return key.

Area	Access	Intrinsics
The main form area	Input/ Output	VSHOWFORM, VREADFIELDS
The Window area	Output only	VPUTWINDOW
The function keys	Output/Input	VREADFIELDS, VSETFKEYLABELS
The menu area	Output	No intrinsics, must be custom coded

Table 3 - Areas of the VPlus screen available for use

What user actions can be defined (what events)?

The number of actions available to a user is simple - pressing the Enter key, or one of the 8 function keys. These 9 events constitute the total repertoire of user actions available to VPlus. All of them are trapped and recognized by VPlus, and passed back to the host program, by means of the Comarea. We will limit our program (VDEMO) to handling these 9 events.

The responses to these events are a little more complex, but not so much so that they become confusing. It is important to remember that we are not in the least concerned with what data is actually *in* any of the fields on the screen. This makes our task easier, and makes certain that we have a truly modeless interface. There are five possibilities for the program's response to a user action:

- Do nothing. The data in the fields of the form may change dramatically, but from the point of view of the user-interface, the screen stays pretty much the same - the form remains the same, the function key labels remain the same, and nothing is written to the window area, or the menu area.
- Stop the program. It is important that we include this option in our prototyping, otherwise we will have to dedicate one function key to the VDEMO program, rather than allowing the prototype model to control this event. This is similar to the way in which ENTRY impeded a full prototyping capability.
- Switch to a new form, with the possibility of using any of the VPlus GETNEXTFORM options. (This includes clearing the screen or appending to the existing form, as well as freezing the existing form onto the screen. These options are documented in the VPLUS manual.)
- Push a specific event into the queue. This is typically done when we have just moved to a new form, but need not necessarily be the case.
- Toggle some character in the function key label. Typically this will be an asterisk, in position 8 of the first line of the label text. However, we may want to make this a little more varied.

How do we store these requirements in a catalog?

Although it is decidedly unscientific, I believe in the power of coincidence. When good coincidences arise (serendipities), I like to take it as a sign that we are on the right track!

To specify a form takes 16 characters, *exactly* the amount of space that it takes to specify the label for a function key-label. The options to VGETNEXTFORM take 2 characters, and specifying the additional response from the program takes another character or two. It seems then, that we can begin to design some message structures for use with our native language catalogs.

DESIGNING THE CATALOG MESSAGES

There are two classes of message used by VDEMO - Form header messages, and Event messages. The underlying structure is of a set of 10 messages. The first message (###0) is a form-header message, which specifies what form is to be used, and how it is to be retrieved. It is also possible to display a message in the window area.

The next 9 messages (###1...###9) correspond to the 8 function keys, and the ENTER key. Each of them includes space for a label (not used for the ENTER key message, ###9), and a set of control characters. This allows us to cause *any* of the responses to occur, since we can code the value for an event in a single numeric character (1..9).

Form header messages

This section describes the layout of a form-header message.

Form ID

Characters 1...16 name the form. This form must be in the form-file specified when the program is launched. If no form with this name can be found, the program displays a message to that effect. If the field is left blank., then whatever form is on the screen at the time is re-displayed.

VGETFORM parameters

Character positions 17 and 18 correspond to the FREEZAP and REPEATAPP fields of the VPlus Comarea. If the character position is left blank, a value of zero is assumed. These fields operate just as described in the VPlus manual.

Window flag

Character position 20 allows a descriptive text to be displayed in the Window area. The text itself is contained in positions 21 ... 80, allowing a maximum of 60 characters. It is important to note that this text is displayed only if the value in character position 20 is a "W". If the field is blank, then the text is used for documentation only, and the previous contents of the Window area are not changed.

Window text

The character positions from 21 ... 80 may be used for either the displayable text, mentioned above, or for purely documentary purposes.

Event (Label key/Enter key) messages

An event message allows VDEMO to take the appropriate action to a user initiated event. (Either the ENTER key, or one of the function keys f1 ... f8). This section describes the layout of the event message.

Label text lines

Character positions 1...16 specify the text to be placed in the corresponding function key label. If the area is blank, then spaces are moved into the label area. This text is displayed each time the form is displayed, which implies that if the value of the text is to change, then a different set of messages must be used. The text is organized as 2 lines of 8 characters each.

Next level

This is taken care of by the NextLevel character, in position 17. Up to 10 levels or sets of messages may be managed by VDEMO. Level 0 is the base level, specified when the program starts up, and level 9 is the highest level If no value is placed in this field, then the same set of messages is used when the form is re-displayed.

Next event

Character position 18 allows the user to specify that a particular event is to be 'stuffed' into the event queue. VDEMO defines 9 events, and any of these events can be stored here, to be used when the function key is pressed. Note that this event may well be an event for a completely different form, since the change of forms precedes the 'stuffing' of the event.

Special options

Character positions 19 & 20 allow further customization of the program's response The special character available here, and their meanings, are as follows:

Pos.20	Meaning
%	Toggle the character value in character position 19 on and off, with successive presses of this function key. The character is assumed to be present in the initial label text. If not, then character position 8 is used as the target location of the toggled character.
*	Exit VDEMO when this event occurs. This takes precedence over *any* new form that may be specified elsewhere in the message.
M	Place the text in positions 21...80 into the menu area. This action takes place *after* any new form has been retrieved. The text is used to replace the function key labels on the terminal. It is cleared by simply pressing RETURN. No function keys are disabled by this action. If this option is not used, then the text in positions 21...80 is used for documentary purposes only.
W	Place the text in positions 21...80 into the window area. This action takes place *after* any new form has been retrieved. This capability is used in Form header messages

Table 4 - Special uses for character position 20 in *event* messages

Replacement label text/comments

Positions 21...80 are used for menu replacement text, or for descriptive comments.

Sample messages from a catalog

Here is a sample of what some messages may look like, when compiled into a catalog.

```
$SET 100 Sample catalog messages
$
0010 DEMOFORMA       00 WThis text will show up in the Window area of the screen
0011    f1              MThis message replaces the function key labels
0012 Go to   Level 2 2   Pressing f2 will move to 'level' 2, messages 20…29
0013                     f3 does nothing, we stay at level 1
0014    f4 *           *%f4 will toggle the * in Label4, line1, position 8
0015                 14  f5 does the same thing - can you figure out why?
0016 Change  Window     WPressing f6 causes this message to appear (1/6)
0017 Refresh         1    A do nothing event, useful in many instances
0018 Exit               *The asterisk in column 20 means Exit VDEMO!
0019                     Nothing happens when you press ENTER!
$
$                        Note how level 2 APPENDS the form
$
0020 DEMOFORMB       10 WThis text will show up in the Window area of the screen
0021 Go to   Level 1 1   Pressing f1 will move to 'level' 1, messages 10…19
0022 Go to   Level 2    WWe are already at level 2!
0023                     f3 does nothing, we stay at level 2
0024    f4 *           *%f4 will toggle the * in Label4, line1, position 8
0025                 16  Guess what will happen - then press the key!
0026 Change  Window     WPressing f6 causes this message to appear (2/6)
0027 Refresh              If no level is specified, stay at the current level
0028 Exit               *The asterisk in column 20 means Exit VDEMO!
0029                 18  In this case we 'force' the Exit event
$
```

These messages are actually supplied as part of the demonstration package.

Running VDEMO

When VDEMO is running, it requires the following pieces of information:

(The values in italics are samples of user input only)

- Formfile to use: *DEMOFORM.PUB*
- Catalog to use: *DEMOCAT.PUB*
- Message Set: *100*
- Base Message: *10*

This information is gathered in a thoroughly modal way, and the program then continues to open the formfile and the catalog. From that point, however, the flow of control is entirely up to the user. All aspects of the user interface are available for testing and demonstration to the user.

Getting hold of VDEMO

SDL/Software is making VDEMO available at no charge to those who are interested in exploring this technique of prototyping a user interface. The program and associated files are available in several ways. First, they will be contributed to the next swap tape at the New Orleans conference. Second, they can be copied onto tape media supplied by you. Last, they can be supplied on a PC or Macintosh diskette. In this instance, Reflection command files, and Session script files will be available to help you upload the files to your system.

SDL/Software reserves all rights to the software, and makes no representations or warranties about the software.

How To Write Quality Pascal Programs

by

Stan Sieler

Allegro Consultants, Inc.
2101 Woodside Road
Redwood City, CA 94062

Voice: (415) 369-2303
Fax: (415) 369-2304
UUNET: sieler@allegro.com

Revised: June 8, 1992

0. Introduction

This paper is concerned with how to write quality Pascal code.

I want the discussions in this paper to cover the four P's of programming: philosophy, performance, problems, and portability. I considered trying to divide the paper into four chapters, one for each of the P's, but realized that for many topics that I would put in one chapter, equally good arguments could be raised for putting it into another one. So, instead of having one chapter per P, I've organized the paper into the chapters: Style, Coding Choices, Performance Problems, and Portability.

In each chapter, the 4 P's are used as the principles underlying a set of guidelines. These suggested rules should not be viewed as though they were the 10 (or %12 or $A) Commandments. They are my rules, the ones that I have used (in various forms for different languages) in the writing of over one million lines of code. Guidelines should grow, as do people. I can look at programs I wrote in 1970 and see changes in style in the intervening years. Despite the changes, the overall look has remained the same.

The primary purpose of this paper is to encourage readers to think about the various aspects of style and form their own coding guidelines. This is an important underpinning to writing quality programs ... in Pascal or any language.

0.1 Throw-away code

Guidelines:

#1-1. There is no such thing as throwaway code. Therefore, your guidelines apply to **everything** you write. Especially the "throwaway" or "one-shot" programs.

Commentary:

#1-1. I've noticed that I almost always re-use my supposedly throwaway or one-shot programs. And, if I've cut corners on using my guidelines I always have to pay for it later.

1. Style

Style, or lack of it, can dramatically affect the quality of a program. Quality code must be written so that it is correct, efficient, and maintainable. Choosing, and following, a particular style can increase the likelihood that each of these three goals are attained. This section will first discuss variables and then the rest of a program.

1.1 Names : The Basics

One of the keys to a successful program is the proper choice of names.

The success of a Pascal program lies in a good understanding of the data the program manipulates. This understanding is reflected in the choice of data structures and variables used by the program.

Throughout this paper the phrase "names", when used without a qualifier, refers to the names of variables, constants, types, procedures, and functions.

Guidelines:

#1-11. Use nouns for variables, constants, and types; use verb phrases for procedures and functions.

#1-12. Types should usually have "`_type`" at the end of their name.

#1-13. Names should be entirely lowercase, with an underscore ("_") separating the words (e.g.: `card_counter`).

Commentary:

#1-11. The following one-line example should suffice to show the importance of this guideline:

```
    if card_count = 52 then
```

Is "card_count" an integer variable or a function? The difference is critical to the understanding of a program. The only guideline the reader can possible have is his/her understanding of English.

It is important to remember that other people will be reading the code someday. ("Other" people includes you a day after you have written the code!) They will need "pointers" to help them correctly interpret the code they are reading.

According to guideline #1-11, "card_count" is a variable. If it was supposed to be a function, then its name should have been something like "count_the_cards".

#1-12. Some programmers prepend "t_", "typ_", or "type_". Others append "_t", "_typ", or "_type". A few misguided souls have been so enamored with this idea that they prepend "t_" **and** append "_type" to every type.

The basic idea is to clearly differentiate the names of user defined types from variables, and procedures (or functions).

#1-13. This guideline reflects the fact that we are English speakers. If We were German Speakers, our Programs should probably use Capital Letters at the Start of every Variable. Notice how much harder to read this is?

I can see three reasons for capitalizing the first letter (or all) of some types of names. None are worthwhile:

- you picked poor names for your variables, and think that capitalization will compensate.
  ...it won't.

- you aren't sure the compiler knows that a name is a variable if it isn't capitalized in a particular style.
  Surprise: the compiler is smarter than you are!

- (mostly for C programmers) you want the ability to thoroughly confuse the reader by having two distinct variables that differ only in the case of their names (e.g.: `int foo; char FOO`).
  ...more honorable paths to job security exist.

1.2 Names : More Details

Names for types should be descriptive, and should be clearly distinguishable as **types** so that they stand out when used in type coercion or "`sizeof`" expressions.

Guidelines:

#1-21. Generic types that are packed arrays of char should be named "`pac`" with the number of characters.

```
type
   pac8     = packed array [1..8] of char;
   pac80    = packed array [1..80] of char;
```

#1-22. Types that are arrays should have "array" in their name. (Exception: pac## type names (see #1-21).)

#1-23. Types that are pointers should have "ptr" in their name.

#1-24. Records should have field names with a prefix that reflects the record they are part of.

#1-25. Simple types should not usually reflect their "size".

#1-26. Try to avoid the use of "`true`" and "`false`" for functional results. Use a type like "`good_failed_type`" instead.

Commentary:

#1-21. The Pascal language has special rules for variables that are packed arrays of char with a lower bound of 1. These special rules were added in an attempt to make standard Pascal usable for text manipulation. As a result, most implementations of Pascal refer to "PAC" as a variable (or type) which is a packed array of char, with lower bound of 1. Hence, it is convenient to reflect this standardization in our own types.

Note that this guideline said **generic** types. When I use TurboIMAGE, and want to create a type that corresponds to an IMAGE item, I will try to give the type a name that is similar to the item's name:

```
   {IMAGE items:                        }
   {   cust_name    x20;                }

  type
   cust_name_type    = packed array [1..20] of char;
```

In the above example, I used "20" in the type declaration only because it was a directly obvious value (given the comment about the IMAGE items). I would **never** use the constant of "20" later in the program. If I needed to refer to "20", I would either use the Pascal/XL construct "`sizeof (cust_name_type)`" or I would add a "`const cust_name_len = 20`" and use that constant to declare the record.

#1-24. This guideline makes the fields of a record immediately identifiable when they are used in a "`with`" statement. Example:

bad:

```
type
   hpe_status      = record{name matches MPE XL's usage}
      info  : shortint;
      subsys       : shortint;
      end;
   hpe_status_ptr_type = ^ hpe_status;
   ...
var
   hs_status_ptr : hpe_status_ptr_type;
...
with hs_status_ptr^ do
   begin
   info := my_info;
   subsys := my_subsys;
   end;
```

good:

```
type
   hpe_status      = record        {name matches MPE XL's usage}
      hs_info      : shortint;
      hs_subsys    : shortint;
      end;
...
with hs_status_ptr^ do
   begin
   hs_info := my_info;
   hs_subsys := my_subsys;
   end;
```

In the *bad* example, the reader who is unfamiliar with the code will not know that the pointer's fields are being changed. In the *good* example, it is obvious.

#1-25. In SPL, it is quite common to use a suffix or prefix to denote the "type" of a variable. (Example: `double ktr'd`). With Pascal's restrictions on matching types this is much less necessary.

#1-26. "`true`" and "`false`" mean little by themselves. Examples:

bad:

```
function open_files : boolean;
   ...
   open_files := false;
...
if open_files then
   ...
```

good:

```
type
   good_failed_type         = (failed, good);

function open_files : good_failed_type;
   ...
   open_files := good;
...
if open_files = failed then
   ...
```

In the *bad* example, the reader has no idea whether or not "`open_files`" returning a true value is good or bad.

This guideline is even more important when programming in a multi-language environment, because different languages (and different operating systems) have strange ideas about whether "0" (or "`false`") means *good* or *bad*.

1.3 Ordering of Source

The ordering of the source code can dramatically affect the readability of a program. Ordering guidelines fall into three areas: types & constants, variables, and procedures & functions.

Guidelines:

#1-31. Simple constants should be put at the start of the declaration area, followed by types, structured constants, and then by variables. Within each group, identifiers should be arranged in some order. If no other order presents itself, alphabetic order should be used. Example:

```
const
   max_queues            = 10;

type
   queue_name_type       = packed array [1..8] of char;
   queue_name_array_type = array [1..max_queues] of queue_name_type;
```

#1-32. At most one instance of "type", and "var" is needed, and up to two instances of "const" (one for simple constants, and one for structured constants).

#1-33. Identifiers in a declaration area (constants, types, variables) should be declared one per line, in some order. Alphabetic order is the default. If another ordering is used, it should be explained with a comment.

#1-34. Identifiers in a declaration area should be entered so that the "=" characters are aligned for constants and types, and the ":" characters are aligned for variables and fields in records.

#1-35. Record types that contain nested records should never be declared as "anonymous types". Examples of bad and good practices are:

bad:
```
type payment_record_type = record
        pay_date : date_type;
        pay_time : time_type;
        pay_style : (paid_cash, paid_check, paid_charge);
        end;
```

good:
```
type
    payment_style_type = (paid_cash, paid_check, paid_charge);

    payment_record_type       = record
        pay_date              : date_type;
        pay_time              : time_type;
        pay_style     : payment_style_type;
        end;
```

#1-36. "Crunched" records should be avoided unless the need for tight packing of the fields overrides the performance loss they cause.

#1-37. Large variables (> 256 bytes) should be declared last, even though this violates guideline #1-33.

#1-38. Procedures and functions are intermixed (i.e.: do not separate procedures from functions).

#1-39. Procedures (and functions) should be declared in some order (alphabetic is the default).

#1-40. More than one level of nested procedure should be avoided.

#1-41. Intrinsics should be declared once, at the outer level, after all constants, types, and variables, and before any "external", "forward", or actual procedures.

#1-42. Intrinsics should be in alphabetic order, arranged by intrinsic files. Example:

```
Function  ascii             : shortint;           intrinsic;
Function  binary            : shortint;           intrinsic;
Procedure quit;                                   intrinsic;
```

In the above example, two spaces were put after the word "Function", so that the names of all intrinsics would be aligned, regardless of whether they were functions or procedures. It is unfortunate that Pascal makes us prove to the compiler that we know the functional type of every intrinsic.

Note the uppercase "P" and "F" in the above example. This is one instance of a very useful coding discipline that is explained in guideline #1-45 below.

#1-43. All procedures and functions should be declared with "forward" declarations, which are in alphabetic order.

#1-44. Types that are "fillers" should be declared as "integer" or "byte" (where "byte" is declared as 0..255) instead of as "char" for their base types.

#1-45. "forward", "external", and "intrinsic" procedure (and function) declarations should capitalize the first letter of "Procedure" and "Function".

Commentary:

#1-35. Variables (and fields of records) that are anonymous types can never be passed by reference (as "var" parameters) to ordinary procedures. Indeed, anonymous records usually cannot be passed by value into procedures.

#1-36. Accessing fields of "crunched" records can take up to three times the number of instructions that would have been required if the record was not crunched. Consider the following two types:

```
type
   bad_type = crunched record
      bad_misc : shortint; {Bytes 0, 1}
      bad_status : integer; {Bytes 2, 3, 4, 5}
      end; {Total size: 6 bytes}

   good_type         = record
      good_misc      : shortint;      {Bytes 0, 1}
      good_status    : integer;       {Bytes 4, 5, 6, 7}
      end; {Total size: 8 bytes}
```

When the "bad_status" field is accessed, Pascal/XL emits three instructions (LDH, LDH, and DEP). When the "good_status" field is accessed, Pascal/XL emits one instruction (LDW).

#1-37. Pascal/XL can efficiently access only the first 8,192 bytes of global or local variables. Consider the following examples:

bad:

```
var
   big_array       : array [0..9999] of integer;  {40,000 bytes}
   ktr      : integer;
```

good:

```
var
   ktr             : integer;
   big_array       : array [0..9999] of integer;
```

In the *bad* example, Pascal/XL will use two instructions to access "ktr". In the *good* example, Pascal/XL will use a single instruction to access "ktr".

#1-38. Pascal's differentiation of functions versus procedures is unfortunate, at best. We shouldn't encourage language designers to perpetuate this flaw.

#1-39. The "$locality" statement can be used to tell the linker to group specified procedures together regardless of their order in the source code.

#1-40. Pascal/XL allows procedures to be declared within procedures that are declared within procedures that are...

Nested procedures pay a run-time performance penalty when they access the variables global to them that are local to the surrounding procedures.

Procedures nested more than a total of two deep (i.e.: an "outer level" procedure and one inner procedure) usually implies that other design problems exist.

#1-43. This extra work often pays off well when writing a "module" that will be linked with other programs. I often put all of my "forward" declarations into a file, and then use the following Qedit commands to generate an "external" declarations file for other modules to $include:

```
t myfile.forward
c "forward"(S)"external"@
k myfile.external
```

#1-44. Debug/XL's Format Virtual command (FV) will produce much more readable output for random data when the base type is numeric instead of character.

#1-45. With this guideline, and an associated one in the next section, a Qedit command like:

```
l "procedure" (s)
```

will list the first line of every procedure declaration. *Note that only the actual declaration will be listed, not the* "`forward`" *or* "`external`" *declarations, since they would have been declared with a capital "P" in* "`Procedure`".

Note: do not tell your editor to automatically upshift all text you search for (e.g.: Set Window (UP) in Qedit), as that will defeat the purpose of this guideline.

1.4 Executable Code

This section discusses the styles of coding for the executable code part of a Pascal program.

Guidelines:

#1-50. All code should be in lower case.

#1-51. Comments should be in English, and should be in mixed case, as is the practice in English.

#1-52. Comments should appear in one of two styles, depending on their size:

- aligned at a chosen column, to the right of code.

- aligned 6 spaces to the right of the current code indentation, with a blank line above and below.

Example:

```
          {the following loop looks for a null character}

     null_index := -1;                    {-1 will mean "not found"}
     test_inx := 0;                               {index of first char}

     done := (len = 0);                   {don't loop if no data}

     while not done do
        begin
              {see if current character is null...}
        if buf [test_inx] = chr (0) then
           begin                          {found a null!}
           null_index := test_inx;                {remember location}
           done := true;                  {terminate loop}
           end
        else
           begin                          {incr inx, check end}
           test_inx := test_inx + 1;
           if test_inx >= len then                {inx is 0-based}
              done := true;
           end;
        end;                              {while not done}
```

#1-53. Multi-line comments may be written with "{" and "}" on every line, or with the "{" and "}" appearing only once, on lines by themselves.

#1-54. The "{" and "}" characters are used to start and terminate comments, never the "(*" and "*)" pairs.

#1-55. "{" is typically not followed by a space, nor is "}" typically preceded by a space (unless it is to align it with the prior line's "}").

#1-56. Lines should be no longer than 72 bytes, even though Pascal/XL allows longer input lines.

#1-57. Blank lines cost nothing at run time, and should be used liberally to separate sections of code. Example:

```
     if ktr > max_ktr then
        max_ktr := ktr;                   {remember new high water}

     done := false;
     while not done do
        begin
        ...
```

#1-58. The "`end`" statement does not have to have a comment on it. Pascal never checks that your comment matches reality, anyway.

#1-59. The basic unit of indentation is 3, not 4 or 2.

#1-60. Indent "`begin`"s 3 spaces more than the start of the preceding line. Code after a "`begin`" (up to and including the "`end`") is at the same level as the "`begin`".

#1-61. Continuation lines are indented 6 more than the start of the first line.

#1-62. The "`then`" of an "`if/then`" statement is usually on the same line as the rest of the boolean expression, not on the next line by itself (unless necessary for spacing, and then it is indented 6), and **never** on the same line as the statement following the "`then`".

#1-63. An "`else if`" construct may be treated as though it were a new Pascal construct: "`elseif`". (I.e.: the "`if`" follows the "`else`" on the same line.)

#1-64. A "`goto 999`" is an acceptable method of branching to the end of a procedure (or function).

#1-65. No other "`goto`"s are necessary.

#1-66. Try to keep procedures below five pages (300 lines).

#1-67. Never use the word "procedure" or "function" in a comment in exactly all lower case. Instead, use "routine" or "PRocedure" or "FUnction".

#1-68. Always terminate a procedure or function with a comment of the form:

```
          end {nameofprocedure proc};
```

#1-69. Put a blank between the name of a procedure/function and the "`(`" of the parameter list.

#1-70. Put a blank after every comma in a parameter list.

#1-71. Put blanks around operators (e.g.: " := ", " + ", " - "), and in front of left brackets (" [").

Commentary:

#1-52. Aligned comments make the code look neater. This practice makes it possible for a reader to easily read the code **or** the comments.

#1-55. The practice of always following a "`{`" with a space and preceding a "`}`" wastes valuable space on the line.

#1-56. Long lines will not list well on most terminals, nor are they acceptable to all editors.

#1-58. I do put a comment on the "end" statement when it is more than about 10 lines from the corresponding "begin".

#1-60. The value of "3" and the injunction against "double indentation" saves space and makes the result more readable. Consider the following two examples:

bad:

```
for i := 1 to 10 do
    begin
        buf [i] := 0;
        foo [i] := 0;
    end;
if ktr = 0 then
    begin
        if not done then
            begin
                ...
```

good:

```
for i := 1 to 10 do
   begin
   buf [i] := 0;
   foo [i] := 0;
   end;

if ktr = 0 then
   begin
   if not done then
      begin
      ...
```

Many Pascal programmers have seen the "double indentation" style because the professor in charge of UCSD Pascal (in the early 1970s) used this style. *Note that he was primarily a teacher, not a programmer.*

#1-61. An example:

```
if (card_counter = max_card_counter) and
      all_cards_accounted_for then
   begin
   ...
```

The goal of indenting continuation lines is to make it clear to the reader that the prior line has continued to the next line. If no extra indentation is used, then it becomes difficult to determine the difference between the **next** statement and the continuation of the **current** statement.

When I have a complex "and/or", I try to make it readable when doing continuation lines:

bad:
```
    if ((card_counter = prior_card_counter)
       and (number_of_cards_left > cards_for_book) ) then
       begin
```

good:
```
    if (       (card_counter = prior_card_counter)
           and (number_of_cards_left > cards_for_book) ) then
       begin
```

In the *bad* example, notice how the "begin" is blurred by the "and" starting in the same column.

#1-62. The word "then" is syntactic sugar: it fattens the listing, and has no redeeming value. When the reader sees an "if", he or she automatically knows that a "then" is coming, eventually. The indentation alone would suffice to tell the reader that the "then" statement has been found. Examples:

bad:
```
    if card_counter = max_card_counter
       then done := true
       else ...
```

good:
```
    if card_counter = max_card_counter then
       done := true
    else
       ...
```

In the above *bad* example, the reader has to mentally sift through the excess verbiage ("then") in front of the "done :=" in order to understand the affects of a "true" boolean expression. In the *good* example, reading the left side of the listing suffices.

#1-63. "else if" constructs are typically found in one of two situations:

- a "run-on" "if/then/else". For this construct, the "else if" on the same line is a natural readability boost.

- a nested "if/then/else". For this construct, the "else" with the "if" indented on the next line is natural.

An example of the "run-on" "`if/then/else`":

```
if token_check ('EXIT') then
   wrapup
else if token_check ('LIST') then
   do_list
else if token_check ('PRINT') then
   do_print
else
   writeln ('Unknown command: ', token);
```

Note: in the above style, I will often put 5 extra spaces before the first "`token_check`", so that a Qedit command like `LIST` *"`token_check`" will show all three "`token_check`" phrases nicely aligned.*

An example of the nested "`if/then/else`":

```
if card_counter = max_card_counter then
   if done then
      ...
   else
      discard_current_card
else
   if current_card = joker then
      try_best_wildcard
   else
      calculate_score;
```

A style I recommend against is the following:

```
if token_check ('EXIT') then
   wrapup
else
if token_check ('LIST') then
   do_list
else
...
```

The above style has drastic readability consequences if an untimely "page eject" occurs in the listing between an "`else`" line and the following "`if`" line.

#1-64. Pascal lacks an "`exit`" statement. Both C and SPL have some form of "return from this procedure right now" statements. This is the only place I use a "`goto`" in Pascal.

#1-67. This guideline means that editor "find" and "list" commands looking for "`procedure`" and "`function`" will never accidentally find comment lines instead. (See also #1-45).

#1-68. This makes it very easy to find the end of any (or all) procedure(s) with a "find" command.

#1-69/70/71. Blanks make code more readable, just as they make English more readable. *Note the blank after the comma in the previous sentence.* Example:

bad:

```
fid:=fopen(filename,3,0);
```

good:

```
fid := fopen (filename, 3, 0);
```

2. Coding Choices

This section deals with choices made in writing executable code.

Guidelines

#2-1. Decide if your style is to have functions that return errors, or procedures that have status parameters (or both), and then stick to it.

#2-2. Don't use "`with`" for simple pointer deferencing. Only use "`with`" if indexing into an array.

#2-3. Try to avoid "`repeat`" loops, using "`while`" loops instead.

#2-4. Try to avoid using "`escape`" outside the scope of a "`try/recover`" block.

#2-5. Avoid "`string`"s in favor of PACs. A PAC is a Packed Array of Char with a lower bound of 1.

#2-6. Use Pascal/XL extensions when possible, unless portability is a primary goal.

#2-7. The "`try/recover`" construct in Pascal/XL is very useful for catching errors: both unexpected and deliberately caused.

#2-8. Use `$type_coercion 'representation'$`. **Never** use the `noncompatible` level of type coercion.

#2-9. The "anyvar" parameter type is useful. Consider using it when you want to pass different types of variables to one procedure.

#2-10. The "uncheckable_anyvar" option for a procedure should be used whenever "anyvar" parameters are declared, unless you specifically want Pascal/XL to pass a hidden "actual size" parameter.

#2-11. When using "anyvar", be sure that the formal parameter type matches the alignment restrictions of the expected actual types. I.e.: if the formal parameter is declared as an "integer", then Pascal/XL will assume that all addresses passed into that parameter are a multiple of 4. Use "char" (or another byte-aligned type) as the formal parameter type if you want to pass any kind of addresses safely.

#2-12. The "default_parms" procedure option should be considered as a means of making long actual parameter lists shorter (for ordinary cases).

Commentary:

#2-1. Sometimes, I return a quick overall result with a "good_failed_type", and a detailed error in a status parameter. Example:

```
function open_files (var status : hpe_status) : good_failed_type;
...
if open_files (status) = failed then
   report_status (status);
```

#2-2. Pascal provides a "with" statement that can, in some instances, provide the compiler with a hint on how to optimize the instructions it emits for your code. Additionally, a "with" statement can save subsequent typing.

An example of a useless "with" is:

```
var
   ptr      : hpe_status_ptr_type;
...
with ptr^ do
   begin
   hs_info := 0;
   hs_subsys := 0;
   end;
```

This is "useless" because the compiler & optimizer would probably have done just as good a job of emitting optimum code if we had said:

```
   ptr^.hs_info := 0;
   ptr^.hs_subsys := 0;
```

Note that the code took five lines using a "`with`" statement, and two lines without it.

Finally, the fact that "`hs_info`" and "`hs_subsys`" are actually fields of the record pointed to by "`ptr`" is somewhat obscured when the "`with`" is used.

An example of a useful "`with`" is:

```
var
   statuses : array [0..9] of hpe_status_type;
...
with statuses [k] do                   {optimize hs_@ fields}
   begin
   hs_info := 0;
   hs_subsys := 0;
   end;
```

I consider this example "useful" because the optimizer would have had more work trying to compile optimum code for the equivalent non-with statements:

```
statuses [k].hs_info := 0;
statuses [k].hs_subsys := 0;
```

In the above examples, it is tempting to align the "`:=`"s with the right-most "`:=`" of the block of assignment statements. I will often do this when I have four or more similar assignments in a row. The advantage is increased readability because we make it obvious that the data is related (because of the aligned "`:=`"s). The disadvantage is that a simple Qedit command designed to list where the "`hs_info`" field is changed (e.g.: `LIST "hs_info :="`) will fail.

#2-3. When a "`repeat`" loop is encountered, the reader won't know what the termination condition is until many more lines of program source are read. This means that he or she will not be able to check for proper setup of the termination condition. A "`while`" loop avoids this problem because the termination condition is clearly specified at the top of the loop.

A repeat loop can usually be changed into a while loop easily:

before:

```
repeat
   begin
   ...
   end
until
   buf [inx] = 0;
```

after:

```
    done := false;
    while not done do
       begin
       ...
       done := (buf [inx] = 0);
       end;             {while not done}
```

#2-4. A "non-local escape" costs thousands of cycles of CPU time to execute. In short, **never** plan on using this construct as a normal method of returning from a procedure. Examples:

bad:

```
    procedure do_work;        {note:  no status parameter!}
       ...
       if problem then
          escape (i_failed);
       ...
```

good:

```
    procedure do_work (var status : hpe_status_type);
       label
          999;
       ...
       if problem then
          begin
          status := my_failure_status;
          goto 999;                     {exit}
          end;
       ...
    999:

       end {do_work proc};
```

#2-5. Strings hide an immense amount of slow and somewhat incorrect code. String concatenation, in particular, can result in "memory leaks" where your process runs out of heap space. PACs are messier to deal with, but much more efficient. This is a performance versus esthetics trade off.

#2-6. Pascal/XL is a very useful language precisely because it has a large body of extensions to standard Pascal. If you eschew using them, you would be better off programming in ANSI C or C++.

#2-7. "`Try/recover`" is **not** guaranteed to catch any errors that occur within it. Unexpected errors (e.g.: invalid index, bad virtual address) invoke an operating system routine called `trap_handler` which will "walk" back through your stack markers looking for the most recent "try/recover" block. This "walk" can fail if your stack is corrupted, and the "try/recover" will not be found. If this happens, and if an appropriate trap handler (e.g.: XCODETRAP) has not been armed, your process will be aborted.

#2-8. Type coercion is one of the best extensions in Pascal/XL. It provides a controlled way of overriding Pascal's type checking. The $type_coercion directive tells Pascal/XL what level of type coercion you want to allow in your program. About five different levels exist. The level I strongly recommend is representation. This level allows an expression of one type to be coerced (treated as) another type if, and only if, the two types are exactly the same size (in units of bits, not bytes).

The noncompatible level tells Pascal/XL that there should be no restrictions on type coercion. This leads to interesting bugs in programs. Some MPE XL system crashes can be traced to using this kind of type coercion incorrectly. The following examples demonstrates how noncompatible can hide errors.

assuming:

```
var
   big_ptr  : globalanyptr;
   my_address  : integer;
   small_ptr   : localanyptr;
```

bad:

```
$type_coercion 'noncompatible'$
my_address := integer (big_ptr);   {will get half of data!}
my_address := integer (small_ptr); {will get 32 bit value}
```

good:

```
$type_coercion 'representation'$
my_address := integer (big_ptr);   {will get syntax error}
my_address := integer (small_ptr); {will get 32 bit value}
```

In the *bad* example, the coercion of big_ptr results in setting my_address to the upper 32 bits of big_ptr (i.e.: the space id), silently losing the bottom 32 bits of the address. In the *good* example, Pascal/XL will generate a syntax error on the attempt to coerce a 64-bit expression (big_ptr) into a 32-bit value (integer).

#2-10. I use "anyvar" often. One example is:

```
procedure zero_var (anyvar foo : char);
      {Purpose: zero every byte in the parameter}
   var
      bytes_left     : integer;
      byte_ptr       : ^char;

   begin

$push, range off$
   byte_ptr := addr (foo);
   bytes_left := sizeof (foo);  {Note:  gets actual size!}

   while bytes_left > 0 do
      begin         {zero one byte}
      byte_ptr^ := chr (0);
      bytes_left := bytes_left - 1;
      byte_ptr := addtopointer (byte_ptr, 1);
      end;
$pop$

   end {zero_var proc};
```

#2-12. The following example, a procedure that most users would call with a "false" in the second parameter, shows the usefulness of "default_parms":

```
const
   print_with_cr    = 0;
   print_without_cr = 1;
procedure print_msg (               msg              : str80;
                                 cr_or_no_cr      : integer)
      option default_parms (cr_or_no_cr      := print_with_cr);
   var
      cctl_val      : shortint;
   begin
   if cr_or_no_cr = print_without_cr then
      cctl_val := octal ('320')
   else
      cctl_val := 0;
   print (msg, -strlen (msg), cctl_val);
         {Note:  ignoring errors from print intrinsic}
   end {print_msg};
...
print_msg ('Starting...', print_without_cr);
print_msg ('Ending');    {does a CR/LF at end}
```

3. Performance Problems

The two biggest performance problems in Pascal/XL programs are using the built-in I/O, and using strings.

Guidelines:

#3-1. Avoid Pascal I/O. Use intrinsics instead.

#3-2. Avoid Pascal I/O. Use intrinsics instead. This is worth saying twice!

#3-3. Avoid strings in performance critical areas.

#3-4. Turn off range checking (`$range off$`) only when you are **sure** your program is running correctly.

#3-5. Use the Pascal/XL optimizer (`$optimize on$`).

Commentary:

#3-1. If you encapsulate your I/O calls, then their underlying implementation can be easily changed to use MPE intrinsics. This also aids portability across operating systems and across languages. The "`print_msg`" procedure in commentary #2-12 is an example.

#3-3. String expressions cause the compiler to emit a lot of calls to "helper" routines. Instead of allocating a single work area in your stack, these routines allocate (and deallocate) many work areas on your heap. This can be an expensive activity, and can lead to loss of heap space, and eventual process termination.

#3-5. If your program runs correctly unoptimized, and has a problem optimized, then there are probably one (or more) uninitialized variables. The second most common problem is using pointers in a way that the optimizer doesn't expect (especially when accessing local variables via pointers).

4. Portability

The portability of programs written in Pascal/XL can be enhanced with several techniques. Keep in mind, though, that most other Pascal implementations are not as rich as Pascal/XL. Turbo Pascal (on IBM PC compatibles) provides some of the same features as Pascal/XL.

#4-1. Avoid these Pascal/XL extensions: `extensible`, `readonly`, `anyvar`, `option`, `uncheckable_anyvar`, `default_parms`, `globalanyptr`.

#4-2. Avoid most $ directives (e.g.: `$extnaddr$`).

#4-3. Use type coercion only for types of identical sizes.

#4-4. Avoid "`crunched`" records. Even most C languages do not have a functional equivalent. This includes avoiding "`$HP3000_16$`".

#4-5. Encapsulate "strange" constructs where possible.

#4-6. Encapsulate I/O calls. This is not a language portability issue so much as an operating system issue.

#4-7. Keep your source code lines short (72 characters or less per line).

#4-8. Use user-defined types like "`int16`" and "`int32`" instead of "`shortint`" or "`integer`".

#4-9. Be aware that fields in records may be packed differently on different machines.

#4-10. Standard Pascal does not allow pointers to point to variables. (They can only point into the heap.)

5. Summary

The shortest summary of this paper is probably: you can judge a book by its cover. If a program looks nice, it probably is nice.

#3107

The CASE for OOP

Linda A. Haake
Speedware Corporation
8300 Boone Blvd.
Suite 500
Vienna VA 22182
703-848-9238

Object Oriented Programming (OOP) began to slip into my awareness about 7 years ago after a Baltimore Washington Area Region Users' Group (BWRUG) talk on Object Oriented COBOL. Since then, we've all been reading and hearing about the concepts, especially in the last two years. We've discussed and used Computer Aided Software Engineering (CASE) even longer. Both concepts and the tools associated with them are meant to improve application system quality through faster development, integration of system parts, and easier maintenance. Can they work together? Can we integrate the two approaches to further improve our applications?

To examine these questions, we first must understand something about Object Orientation (OO) and CASE. Then we need to look at how the two can interact or support each other. Because a CASE tool provides a development environment and produces an application, there are actually two ways to look at its object orientation - in the development environment itself, and in how the tool supports the creation of an object oriented application system. Because this paper concentrates on the actual production of an application, the focus and examples are of a lower CASE tool.

Object Orientation is a theory or high-level way of organizing a system, whether that system is a series of processes within a company, a database, or an application. Gordon McLachlan "quotes" Dr. Perkin T. Fudd saying "All systems are object-oriented, whether one thinks so or not." (1) In other words, since people think in terms of objects, our thinking is reflected in our systems. Some theorists believe that building object oriented systems requires a specifically object oriented language. Since OO is a theory, however, others show that the principles can be applied using existing languages and tools. A CASE tool can be one of those.

Some of the concepts of OO are already familiar to programers and analysts under other names. The key concepts are ABSTRACTION and REUSEABILITY. By STANDARDizing, CENTRALizing, and MODULARizing, a developer can create better abstraction and more reuseable objects. The definitions of OBJECTS, their METHODS, CLASSES, INHERITANCE, INSTANCES, MESSAGES, ENCAPSULATION, and POLYMORPHISM provide the structure. Object Orientation looks at a world of interacting objects, figures out what they are (attributes), what they do (methods), and what they say to each other (messages). The art of OOP is defining the objects and their limits.

Early computer systems were programed very directly, by changing their wiring or flipping switches on the hardware itself. Assembly programming languages were another step, but belonged to particular machines and required knowledge of the hardware's memory registers and other specifics. The advent of COBOL and other third generation languages raised the level of abstraction, and allowed programmers to manipulate symbols by using words. Fourth generation languages are even more abstract, removing the need to specify the data access or the procedural order. Object Orientation is the next step in this process. The purpose is to allow developers to concentrate on the goals of the system and spend less time on the storage of data and the detailed behavior of the interacting parts (objects).

Reusability implies abstraction. When we use an object - a program, a user definition, data access statement, or other part - we expect it to behave in the same way. We shouldn't need to know that allowing the user JOHN to look at some information means that the user object applies its security methods by checking the session name on an HP3000, the login name on an HP9000, or asking for a password on a DOS machine. All we need is the abstract idea that if putting JOHN (or anyone else) on a list for that object lets him see data.

Objects are perceptible items differentiated by their characteristics and behaviors. OOP objects are the abstract representation of members of a CLASS or group of like items. When we think about people or animals or computers or desks, we don't separate their behaviors from their physical characteristics. For example, we may describe Mary as having brown hair, glasses, being reliable and loyal. Physical and behavioral information applies. This is how OO 'objects' are defined - by their behaviors (or METHODS) and characteristics (or DATA). It's an important difference from classic application development, where data is usually thought of as separate from its behavior, and programs are thought of as active segments that do not contain data attributes. Using our example of a user object, both the data - JOHN - and the method - security checking - are parts of the object.

CLASSification, or the grouping of objects that share at least one behavior or characteristic, is another way of providing abstraction. Creating and collecting related objects helps institute standardization and centralization. The fact that classes of objects behave in a similar way or have similar attributes means we can understand and use them without knowing each object in depth. In an object oriented system, objects INHERIT the methods and data of objects which are hierarchically above them in the same class, and can pass them to other objects in subclasses below. For example, all programs in an application may share the characteristics of having names and transforming data. The screen programs inherit these characteristics. They also have names and transform data, and they have fields which can receive input from a mouse or keyboard. One particular screen program may have fields with labels and data describing an employee. That screen retains original characteristics and behaviors of the members of the class of all programs, and has some of its own.

The particular screen occurs one time, and thus is known as an INSTANCE of the class of all programs. An instance is a specific occurrence of an abstract object. In OOP, methods act on instances and actual characteristics belong to an instance.

In order to be reusable, objects must be INDEPENDENT. They rely on their own methods and attributes, and ones inherited from members of their class, to perform their functions. This independence, or encapsulation, means an object can be removed or added for a particular use without creating a ripple effect, requiring changes in other areas, or requiring other objects. It's another way of expressing the Object Oriented concept of modularization.

Independent objects must still communicate. MESSAGES are the requests passed between objects, asking an object to change its state or return information. Using OOP terms, a message requests that method(s) be applied to instance(s). Messages do not have characteristics or methods of their own.

Finally, the principle of POLYMORPHISM applies when we create objects. Conceptually related methods share the same names or symbols across or within objects. The simplest way to understand it is to look at the "+" symbol in many languages. The following two statements share the symbol and concept, but with different actions at the detail level.

```
     LET INTEGER-SUM = INTEGER-A + INTEGER-B;
     LET WHOLE-STRING = STRING-A + STRING-B;
```

Conceptually, we've asked to ADD the first parameter to the second. When the language does the work, it decides to mathematically add the integers and return an integer answer for the first statement, or to concatenate the strings and return one string for the second. Because polymorphism is at work here, we only have to know we want to add two parameters. Abstraction and its benefits are supported again.

Polymorphism can exist at different levels, and is one of the keys to REUSEABILITY. When an object performs in a similar way at a higher conceptual level regardless of parameters given it, but somewhat differently at the detail level, we have a polymorphic object.

The Object Oriented principles just presented are not all unusual. Many of us employ some of them in any of our systems. Including both attributes and behaviors in an object is probably one of the key differences in vision. But it's important to remember that ALL of these principles must be present and used consistently if an application is to be considered Object Oriented, and for the benefits to be realized.

We can define CASE tools more quickly. They also depend on abstraction, centralization, and standardization to bring their benefits to development. The focus is to allow a developer to shorten the time needed and improve the reliability of application systems.

At least two levels of CASE tool are typically discussed, although some refer to three. Upper CASE tools provide a structure for the conceptual definition of systems, concentrating on defining the major entities and relationships, and the planning and requirements phases of development. Most generate some kind of data dictionary and may create simple screen and report designs. Other design criteria may be produced here too. Lower CASE tools take up the process at the literal design phase, and focus on the production of the programs and generation of code necessary to build a complete application. Integrated CASE (I-CASE) tools are those which cover both levels. Few of these are available or complete today in the HP world, although these tools are being developed rapidly. Because we're interested in the creation of objects in this discussion, the examples and focus will be on a lower CASE tool.

Applying Object Oriented principles in a CASE environment is a thinking process first. Some of the drawbacks of implementing a new Object Oriented language like Smalltalk or C++ may be as important as the desire to add CASE tool benefits. The learning curve; the necessity to build, buy, or borrow object libraries; and the difficulty of integrating new languages with existing systems can add to the costs of implementing a new language. A good CASE tool, because it should provide an integrated approach, a central repository including the basic structure of many objects, and by nature an abstract view, can provide a faster implementation.

The relative 'openness' of the CASE tool should also be considered, however. A tool that requires us to use its own design methodology probably isn't adaptable to an object oriented approach unless that is its method. Some are poorly integrated across functions, for example, using different syntax for different structures. A tool that provides only methods for defining and generating screens, reports, and batch processes may be too limited for an object oriented approach, since it separates attributes and behaviors, and ignores the need for menus, security, and documentation. Some CASE tools do not allow the user to call other processes, utilities, or subroutines. Others do not allow detailed calculations and comparisons, or don't allow procedural code to be stored within the repository, requiring generated code to be released from the centralized, integrated environment. If a 4GL is generated by the CASE tool, be sure the language is also flexible, capable, and performant.

How might we go about doing object oriented programming within a CASE tool? The following discussion relates the OO principles to examples of creating objects within Designer, Speedware's CASE tool. The same ideas would apply to other CASE tools that meet the criteria of openness, flexibility, and integration discussed above.

As we stated earlier, the object orientation of a CASE development project can be discussed at two levels: that of the CASE tool itself and the environment it provides, and the application being developed. Speedware was designed in an object oriented manner. Both the underlying 4GL and Designer exemplify standardization, modularization, and centralization. Abstraction through hiding access, providing formatters, and structuring classes of objects is a driving force behind the tool. The repeated use of common syntax and object structures illustrates their polymorphism and reusability. Designer's repository centralizes all the necessary objects and utilities in one environment. The consistency, integration, and repetition shorten the learning curve considerably.

We are, however, more interested in developing objects for an application. Here are some examples of objects and how they employ OO principles.

Identifying objects, their classes, and their limitations is a key function in developing an object oriented application. Recalling that it's natural for us to think in terms of objects, let's look again at our user, JOHN. JOHN is actually an INSTANCE of a user, which is our first object. That object may actually belong to a superclass of users who share certain capabilities. To make the actual objects that represent this concept, we would create a user team object in Designer, that has characteristics such as a name and title, and behaviors such as check the session name on MPE-XL and the login name on HP-UX. Fortunately for us, those methods are already built in to the basic user object in the tool. Then we add a user object called JOHN, and make JOHN a member of the higher level team, so that he can INHERIT the capabilities of the CLASS he now belongs to. We may add other characteristics or methods to the JOHN object, or we can just allow whatever is needed to be inherited from the higher level. (See Diagram 1).

Throughout our application, whenever we refer to the object JOHN, all the characteristics and behaviors come along. We may create another user named MARY within the same team or class, but we may also with to limit the hours MARY can work. (Diagram 2). The MARY object then inherits the original characteristics and behaviors, and adds some of her own.

Another example of classes and inheritance can be described by creating another object. Suppose we have determined several reports in our application need prompting screens to collect information about what version or criteria to use for the reports. Rather than make different prompt screens for each, we can make a single, polymorphic screen object that can be REUSED. At the higher, application level, we have set the default size, shape, colors, error messages, and other characteristics and behaviors of screens. When we create the screen, we can create it as a member of the class of programs that belong to our application, so it inherits those attributes and behaviors. Programs are actually a subclass of objects belonging to the application, and screens are a subclass of programs. So the screen we create will also inherit characteristics and behaviors of the class of objects called programs, and the subclass called screens. In plain English, the developer does not have to write or specify anything about how the object interacts with the 'output file' (terminal or monitor), the data, the size or colors, a keyboard or mouse, the position of titles, function keys, modes, or any of a number of other items that determine how the new object actually works. Again, where we want a particular object to be different from it's higher level, we can change the attribute or behavior for that particular object without disturbing the inheritance of other parts.

We do want our prompt screen to work with different reports. For example, all 4 of our model reports can be run to include all data or data for a range of dates. Three of them can be sorted either by customer or state, and one can be sorted by customer, or region, or state. Obviously, we wouldn't want the users to have to remember these details, and we want the screen to be INDEPENDENT of other objects and reusable. Remembering that POLYMORPHOUS objects behave differently according to what parameters are passed, we realize we will have to build the screen object so that it reacts to parameters.

The fields needed are All Data, Beginning Date, Ending Date, Customer Sort, State Sort, and Region Sort. All Data, Beginning and Ending Date are needed by all the reports, so we can place them on our screen without worrying. We will add logic that prevents anyone from requesting both All Dates and a Beginning and Ending range, as well as insuring the Ending is greater than the Beginning, and that all the dates are valid (no 02/29/91 for example!). If Beginning and Ending Dates are defined as Date type fields, they retain the behavior of date fields, which will automatically insure against invalid dates.

What about the sort fields? They appear only under certain circumstances. By replacing the display and modify options of these fields with parameters, we can control when they appear. We'll probably want to supply default values for the parameters too. Similarly, we will use a parameter to determine which report to run after the prompt screen is used. (Diagram 3).

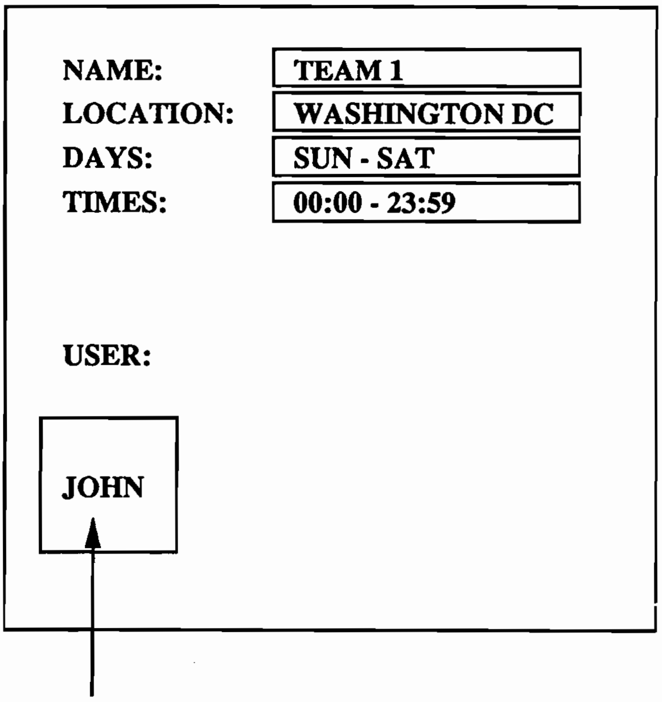
The parameters help maintain this object's encapsulation, or INDEPENDENCE. The parameters are the MESSAGES sent to and from the object. Each time we use the object, we pass it the parameters that say 'turn on' or 'turn off' to the sort fields, and the name of the report. The screen passes its parameters, or messages, that say 'all dates', a date range, and/or the sort information to the proper report. To use the object again, we simply pass the parameters needed. The screen object does not depend on another object to do its job. When we need to use a similar object in a new situation, we can REUSE it. Should we need to collect another sort item for the next situation, it could be added without changing the existing fields and parameters.

These are fairly simple examples of applying object oriented principles to development with a CASE tool. The CASE tool used for the examples is structured to provide the integration, flexibility, and openness we need to apply the OO principles. By preserving the abstraction, independence, and polymorphism of objects; relying on the consistency and time saving supplied by using classes and inheritance; and utilizing the abstraction, centralization, and integration of a good CASE tool, the benefits of both are realized.

FOOTNOTES

1. HP Professional, May 1992. McLachlan, Gordon. "Lessons in Object Orientation", p. 42.

CLASS OF ALL USERS



CLASS MEMBER

DIAGRAM 1

CLASS OF ALL USERS

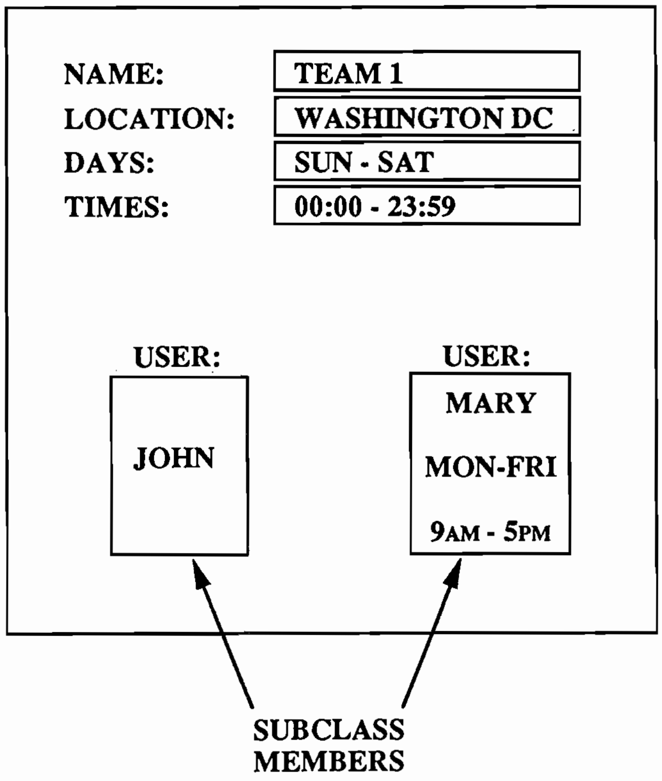


DIAGRAM 2

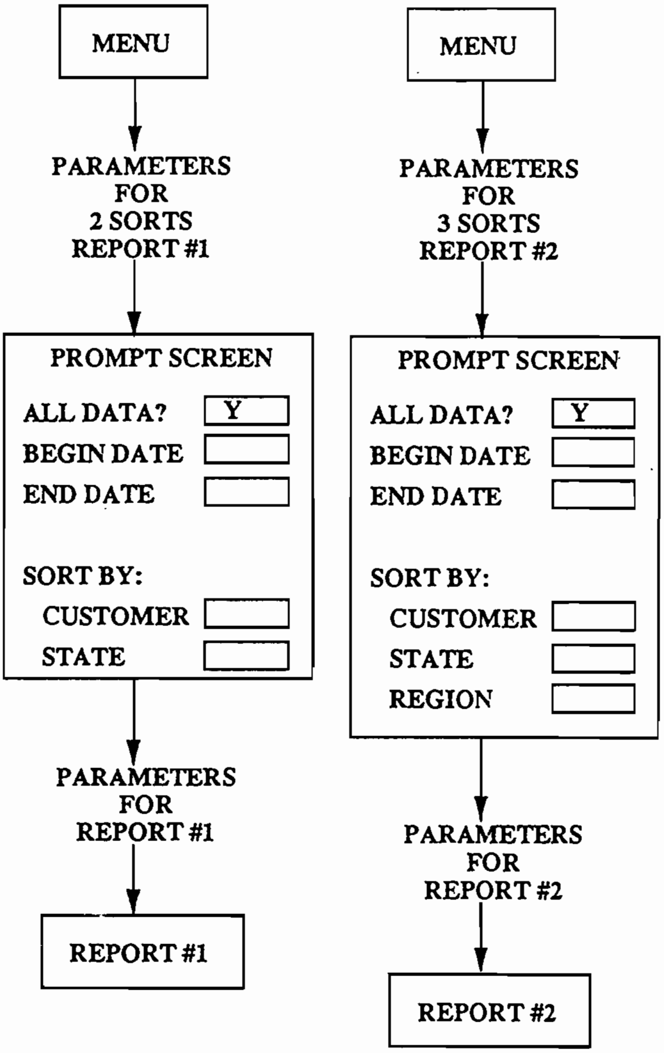

MENU
PARAMETERS FOR 2 SORTS REPORT #1
PROMPT SCREEN
ALL DATA? Y
BEGIN DATE
END DATE
SORT BY:
CUSTOMER
STATE
PARAMETERS FOR REPORT #1
REPORT #1
MENU
PARAMETERS FOR 3 SORTS REPORT #2
PROMPT SCREEN
ALL DATA? Y
BEGIN DATE
END DATE
SORT BY:
CUSTOMER
STATE
REGION
PARAMETERS FOR REPORT #2
REPORT #2

DIAGRAM 3

Paper number: 3108

The Object is the Subject

Douglas Colter

Speedware Corporation
150 John Street, 10th Floor
Toronto, Ontario,
Canada M5V 3C3
(416) 408-2880

Object-oriented programming is beginning to emerge as one of the most talked about development methods of the 1990's. However, object technology can only be successfully applied to the software development environment if it is approached with a good strategy.

Object technology is more than just a programming language. It is a strategy which involves programmers, DB administrators, analysts, and management. This paper discusses the basics of object technology (OT), and then an approach to the successful implementation and ongoing use of object technology.

Will object technology rid you of your woeful programming situation? Is object technology just an other passing fad that keeps technical writers employed by inventing new acronyms, or is it possible that this new technology can deliver its promise in addressing today's very complex problem domain?

First of all, is there even a need to introduce new software development tools and methods? After all, it took over twenty years to define structured programming tools and methods and finally convince some people to use these techniques. Going object-oriented requires a "leap of faith", but some people "still prefer to remain agnostic." [1] The fact remains that there are still an estimated seventy billion lines of COBOL code in use. [2] MIS cannot abandon these monolithic legacies, and the cost of rewriting them would be astronomical.

Programming tools have been evolving since the days of assembly language. There has always been a reluctance to shift to new development tools after investing in current ones. The reason is the time required to master new tools and methods are lengthy and sometimes expensive. Often MIS feel they cannot wait to learn new technologies when the required applications must be delivered immediately. However the complexity of the problem domain has been growing at an increasing rate. This complexity is often compounded by the chosen software tools and methodologies.

Conventional design methods cannot easily accommodate today's user demands for fast delivery of high quality software. Many companies looking for new development technologies to address these concerns have turned to object technology. James Martin of James Martin Associates said "More advanced organizations are modelling their enterprise in terms of objects. By the year 2000, the advanced corporation is going to have a very thorough object-oriented model." [3]

The introduction of the object technology paradigm is more than a simple matter of learning a new language. It is a strategy involving the use of new development tools, new analysis and design methods, different ways of project estimating, and most important, user acceptance. Success requires ongoing training and education for all people involved in developing or using object technology.

Object-oriented Programming (OOP) is a buzzword that has been around for several years. OOP tools have been in use for a little while but they still have a long way to go before they ever becomes widely accepted. It is important for developers to realize that OOP is not an end in itself, but a means: to provide developers with a way to easily develop and maintain very high quality software that is more functional and easy to use by end users.

This goal is one that every developer dreams of. Although OOP tools continue to emerge in the commercial market place at a fast rate, object orientation is not something you can easily implement for the first time. Object-oriented systems must be designed with great care. And it is difficult to do it right, technically. Understanding object technology is definitely a prerequisite to using tools that support it. Also since many of the tools and methods are still in a state of evolution the onus is on the developer (rather than the tool) to manage this programming environment.

In order to make object technology available to a broader range of people programmers, designers, and management must be educated further. For example, technical managers need to be aware of the impact of object technology on project planning, application designers must learn new analysis and design methods, and programmers have to be trained and equipped with tools that support the technology.

What are some manifestations of object technology?

There are at least three different implementations of object-oriented technology available. One implementation of object technology is found in the end user interface. A GUI is used to present data or methods in the form of pictures. An example is where the user drags a picture of a file folder to a picture of a trash can. Any programming language that has access to the window or panel interface could have been used to achieve this affect.

A second form of object technology is found in an object-oriented DBMS. An OODBMS permits the storage and retrieval of various kinds of objects. An example of the kind of data stored is a binary-large-object (blob) which might be a picture of a house in a real estate system. These databases are not widely distributed because they are highly specialized and often cost upwards of $100,000. Also, it is not necessary to strictly apply object-oriented techniques to use an OODBMS.

Object-oriented programming languages represent the third form of OT software. Although these languages support the definition and manipulation of objects, they usually don't fully manage or enforce OT concepts. A programmer still requires a good understanding of the underlying technology and a good design and implementation strategy.

What does object technology provide?

The most important benefits from using objects is increased productivity and improved software quality. Object based development tools provide formal support for programmers to create and maintain reusable components. By defining a formal approach to building objects, programmers can reuse other programmer's code, rather than reinventing the wheel each time. Using a formal approach to objects, a programmer should never be concerned with the processing details of an other object, but rather what service that object will deliver.

Once programmers accept the idea of reuse, it becomes economically attractive to spend more time improving the quality of objects. The increased quality of objects leads to further code reuse and also improved quality in software applications. If an application can be built using well defined objects, then the cost of keeping the application up to date over time is greatly reduced.

With all these promises of increased productivity and quality let's begin with a quick lesson on object technology basics. Object-oriented programming is a new approach to problem solving often using new programming tools (the paradigm shift) where problems are modeled through the manipulation of objects.

Objects

Objects are software abstractions of physical or conceptual entities found in the real world. Objects are often described as self-contained packages of data structures and functionality. Data abstraction supports the manipulation of data by objects through a generic interface between objects. Objects may contain other objects as *instance variables*. For example a screen program format object may contain a field definition object. You add functionality by defining behaviours (*methods*) for objects. Methods and data formats (data abstraction) are usually hidden by the object.

Encapsulation

Encapsulation is the concept of hiding the methods contained in objects. An example may be found in powerful fourth generation languages (4GL) where a screen program definition handles the modes of data manipulation, methods of user interaction, file access, and procedures. In this example the 4GL engine is responsible for many of these processes. Objects handle the processing details, responding to messages by applying built-in methods appropriate to their data. The complexity of software applications is reduced because objects allow the programmer to be concerned about *what* they want rather than *how* to get it. The concept of encapsulation is important from a software maintenance perspective because the changes made to methods are localized to within objects. Changes are applied to the objects without having to change the interface between objects.

Messages

In object-oriented systems you perform operations by sending *messages* to objects. Messages are received by objects in the form of parameters. These parameters are invariants which are often defined in a way so that different objects can respond to the same generic message in different ways. A simple example would be to send a customer number to a report program which displays open orders for a customer. The same message (customer number) may be sent to an orders screen where orders may be entered for that customer.

Classes

In an object technology environment, all objects belong to *classes*. A class is like a template providing a common way to define the type of object where one or more objects share the same set of attributes and services. Classes are organized as multiple hierarchies or subclasses where the methods of a subclass are *inherited* from a superclass. Using subclasses, new objects are defined as incremental modifications to existing objects.

Inheritance

Inheritance lets you define subclasses that automatically *inherit* the characteristics of their superclasses. Inheritance allows for common abstractions to be represented uniquely. An example would be the data entity relationship model: A database is composed of files; a file contains records which are composed of items; and an item would be an occurrence of an element. In this example, the item object will inherit all attributes defined in the element object such that the item would be *like* the element. However, attributes at the item level can override inherited attributes providing subtle distinctions between objects in the same class. With this design approach the client (item) of an old service (element) doesn't have to be modified to use the new service. In the same example, you can make changes to the rules or methods of the element and the items that use the service of the element will automatically *inherit* these attributes. Inheritance allows reuse of objects and their interfaces. It also supports *polymorphism*, where operations may be applied to objects in the same class of objects.

Polymorphism

Using the same message to communicate with different object types is known as *polymorphism*. Polymorphism simplifies programming because different objects sharing a common interface can be manipulated by sending generic messages to those objects. An example might be to send the same customer number to an object which has been defined to return the current value of unpaid invoices. It is not necessary for the calling program to know how the invoice total is calculated, nor is it necessary to know whether the invoices are stored in a network database or a relational database.

Object-oriented tools

Now that some of the basic object technology concepts have been discussed, we need to identify some tools that support this technology. Objective-C and many flavours of C++ represent a new generation of languages which have evolved from C. Both of these languages provide syntax to define objects and classes which allow you to reuse them at any level within an application. Smalltalk, Eiffel, and Object Pascal are other rising stars in the object tools world. All of these tools run on either UNIX or DOS and usually run in a GUI workstation environment.

"Despite their benefits OOP and C++ have their pitfalls" says Bill Gates of Microsoft Corporation. [4] "Not only is C++ difficult to learn, but the sophistication and high level of abstraction of C++ code make it even more difficult to design, debug, read, and navigate..."

Object-oriented COBOL is slowly entering the marketplace. Object programming was introduced to the CODASYL COBOL Committee in May 1989 by Megan Adams of Hewlett-Packard. [2] Also, at that time Realia and Micro Focus were developing OO extensions to COBOL. Microsoft will soon unveil their DOS based version as well. The objective of OO-COBOL is to "provide a sophisticated modelling capability... to handle real world complexity..." said Megan Adams and Dmitry Lenkov of Hewlett-Packard. [5] They also said that maintenance should be eased; software reuse must become easier; and finally, considering the estimated 70+ billion lines of COBOL code, "OO-COBOL must be designed as an extension of the existing language rather than a new language."

Object-oriented programming is by its very nature interactive and exploratory and requires tools to support these methods of development. Therefore, I believe, you need higher level tools if you ever hope to define and reference the objects that eventually become the building blocks for your application. Two such object-oriented 'middle' CASE tools are available today: Speedware/ Designer from Speedware Corporation is an object-oriented interactive programming environment (IPE) used to generate object-oriented 4GL applications for the HP3000, HP9000, and other platforms; and Rational Rose from Rational Consulting which is used for prototyping and development of C++ class libraries for Sun and RS/6000 operating systems.

Traditional versus object-oriented program development

In the traditional structured programming development environment, the concept of functional decomposition is used. Typically this approach puts an emphasis on the methods needed to solve a problem with little regard to the representation of data structures. Using structured programming techniques the developer views a program or module as a snapshot of a particular procedure at a given point in time. If a change occurs in the data model for example, one or many modules will become obsolete until they have been modified to reflect the change. In other words a simple change to one part of a design may have a ripple effect throughout the design.

Object-oriented programming is perhaps an evolution of creating structured modules. The structured module is designed to take care of a specific task. Therefore software reuse is not easily accomplished on a large scale. An object, on the other hand, is like a hybrid of the module which encapsulates both data and process in a way that generic calls (polymorphic) can access them.

Although 3GL tools such C++ and OO-COBOL support object-oriented development, a CASE tool which uses a repository to store all objects for all applications is essential when the designs become large and complex. Object-oriented CASE tools allow the developer to create objects in a standard and consistent way where encapsulation and inheritance become automatic. Objects may also be categorized by class so that they can be accessed generically by object browsers. In a large-scale enterprise modeled with objects, the only way programmers are going to reuse other programmer's code is if there is an easy and consistent way to find the required objects. Another benefit of object-oriented CASE is that all non procedural syntax will be generated by the tool. Even in powerful 4GL's there can be a lot of tedious coding.

Even though a powerful object-oriented CASE tool can assist the developer in reducing the time to design and generate objects, there is still no substitute for solid requirements analysis and good system design. The key to designing an application that will survive years of evolution is to build high quality reusable objects. The trick then is to apply all the development techniques you have used in the past. However these techniques should be applied at the class level as well as at the application level. Using these techniques allow you to reuse not just the object code but all the analysis, design, testing, and maintenance that went into each object as you traverse the various class levels.

What design methods should be used with object technology?

Analysis and design should be done in stages, in conjunction with development of the application. Traditionally, the long and arduous process of analysis and design was completed on paper before any coding started. With strong repository based object-oriented CASE tools the design and development phases occur simultaneously through prototyping. The design of a business model should be more of a sketch than a detailed description of the application. The model is represented by interacting objects that reflect real-world business objects and activities. Prototyping permits the developer to create a simple representation of the model without concern for details. Tests of the prototype quickly reveal any design flaws in the model. Once there is consensus on the design of the model, detailed work may begin at the next level.

Defining object classes helps the developer identify requirement details for objects. At this level the methods and variables used in objects may be determined and coded. What is interesting about such an incremental method of development is the design at one level of objects within a class represents the analysis for the next lower level.

The very nature of objects having encapsulated data and functionality allows for changes to be applied very easily. In a business world where change is the only constant, the inherent flexibility of objects also addresses the need to build very complex applications. Using prototyping the development of layered components lets the developer and users see the application without generating rhemes of specifications. The overall design time will get the application delivered sooner and cheaper.

Let's take a look at the design phase in more detail. The design phase begins by identifying the major objects within the business model. The model may represent the entire business enterprise, or perhaps activities in the business such as a financial system. The model seldom changes, but the methods and variables within the model do. For example, in a billing and receiving system there is always a flow of information coming from the order entry process, customers are billed, and some form of payment is received. Exactly how the order is made or the billing calculated is not important at this level. The design of this system can be sketched out as a model consisting of objects that reflect the activities and events which occur in the practice of billing and receiving.

As the model is developed, we can document the role of each object and its responsibilities within the model. Therefore we can say that design occurs at the level of the model and requirements analysis at the object level. It is not necessary to be concerned with the details of the method used within the objects until later.

Once the model is complete the developer can refine the object classes. Specific methods and variables are determined and coded for each object within the class. Because the model changes very little over time it is independent of how details are implemented within the objects. As the requirements of individual objects change over time, their methods may be changed without disturbing the model.

Object-oriented programming supports and encourages component level development where encapsulated objects can only be accessed through well defined interfaces. Using the interactive programming environment found in object-oriented CASE, the requirements, designing, and testing is completed for each individual object. Unit testing allows the developer to find problems early in the development phase when the cost of fixing defects is small. The object interfaces can be tested by assembling them into functional modules. Again, because of encapsulation, objects and interfaces can easily be tested even if the methods for some of the objects haven't been determined. The developer can provide reasonable values for any incomplete objects.

The design phase is an iterative process. During this phase the model is refined by cycling up and down the various levels within the design model adding detailed functionality and refining the methods. Once the objects are completed, the developer can roll back up the process and assemble the model. When the model is tested as an integrated unit, the developer can see what is missing and determine what refinements must be made to the objects, classes and messages. The development, testing, and implementation of the application goes by quickly because these phases are reused whenever objects are reused. This approach is based on continual elaboration of the details of the various classes, objects, and their relationships, rather than on the more traditional waterfall method, with its clear transitions between analysis, design, and implementation. [6] This process emphasizes identification of reusable classes and objects, as well as derivation of new classes from existing ones.

A requirement in the development cycle is the need to work directly with the individual classes and objects, independent of any particular application program in which the components are to be used. According to Sesha Pratap of CenterLine Software [7] the traditional unit of software is the program file or collection of files linked together. As a result, today's development tools such as text editors have been designed around static program files. An interactive programming environment (IPE) such as the one provided by CASE tools let the developer work directly at the component level. The IPE allows the developer to design, test, and maintain components whether or not they are part of a complete program. When the application is broken down into its fundamental components the developer can create and test each piece individually, or integrated with other components. An object repository based IPE also "minimizes the relevance of file boundaries and makes it easy for programmers to work at the component level of abstraction" says Pratap.

The primary benefits of object technology is reuse. The developer must be able to find and comprehend objects at the component level. The IPE should provide browsers for instance that find and display the attributes of the components. Also, the IPE should provide a convenient way to document components to facilitate browsing and object identification. In summary, Pratap says "a complete IPE must fully support the interactive development of O-O code, comprehension and use of reusable components, and the production of high-quality software."

Managing object-oriented projects

Understanding the dependencies and the variables that affect a project schedule is, of course, the essence of project management. It will probably take some time before project managers gain the experience required to estimate object-oriented projects when it comes to factoring in code reuse, prototyping, and other elements which affect schedules. Initial efforts are not likely to show substantial gains since there is a steep learning curve in an environment where the methods are still immature. Practice in defining classes and reusing objects is the only way project managers will be able to estimate and manage object-oriented applications and reap the benefits of OT.

Conclusions

The incremental, iterative style of object-oriented development requires changes in the way projects are managed. The structured paradigm of software development supports limited forms of code reuse, but it doesn't support larger-scale reuse adequately. The reason is because this paradigm doesn't support the notation of multiple hierarchies of abstractions.

Getting the benefits from OT requires shifting from code-based development to component-based development. Furthermore, developers must shift to a higher level of abstraction. For instance, developers must put much more emphasis on analysis and design at the component level. This approach means applications are built by assembling components, much in the same way buildings are assembled using prefabricated sections.

Although the definition of object technology and the tools which support this paradigm is still being refined, it is definitely not too early to get started. Work should begin now. Get started by educating all people involved. Change does not always have to hurt. I believe we have focused too much on the desired result, not the process by which we get there. Remember that tools and methodologies should be chosen to help you think, be creative, and productive. If the tools you use don't, it is time for a change.

Bibliography

[1] Bob Petrovic. Speedware Corporation, April, 1992

[2] Ken Belcher. *Object orientation - the COBOL approach*, OBJECT Magazine, May/June, 1991

[3] Jessica Keyes. *Code trapped between legacy, object worlds*, Software magazine, June 1992

[4] Bill Gates. *Microsoft and object-oriented programming*, OBJECT Magazine, Mar/Apr, 1992

[5] Megan Adams & Dmitry Lenkov. *Object-oriented COBOL*, OBJECT Magazine, Mar/Apr, 1992

[6] Anthony I. Wasserman. *Object-oriented thinking*, OBJECT Magazine, Sept/Oct, 1991

[7] Sesha Pratrap. *Achieving component-bases programming today*, OBJECT Magazine, Jan/Feb, 1992

Paper # 3110

Why is "EDI Standard" an Oxymoron ?

Trevor Richards
M.B. Foster Associates
9417 Great Hills Trail
Suite 2038
Austin, TX 78759
(512) 345 5376

1. Introduction

An oxymoron is a pair of words which are commonly used together, but which can be interpreted as being either complimentary or conflicting terms. For instance, "airline food" can be considered an oxymoron, as "food" is hardly an accurate description of the "nuked" offerings which are served by airlines today.

The objective of employing EDI standards in the representation of business documents in an electronic form, is to minimize the variety of different formats of a particular business document that any given company has to handle. The EDI standards setting process is invariably handled by committees, and as a result the transaction formats defined, in the ANSI X12 standard in particular, allow for each trading partner to demand that different subsets of the total data structure are used.

This paper will consider the types of variation which can occur between transactions from different trading partners, and whether this variety should be handled by EDI software packages or by application interface programs. Initiatives by specific industry action groups and an ANSI X12 working party to address this problem will also be discussed.

When data is received from a customer or vendor in an EDI standard format, the task of restructuring this data, into a format that is more readily interfaced with in-house business systems, is generally handled by a third party EDI software package. One of the main reasons for using such a package is that the data structure employed in transactions which have been formatted to comply with the popular EDI standards is unlike any other data structure in common usage within business applications.

2. Bluffer's guide to ANSI X12 EDI data structures

a. Documents and Envelopes

Business documents, such as purchase orders, have traditionally been placed in envelopes carrying the appropriate address to send them to the appropriate recipient via US Mail, or some other carrier. The EDI equivalent of a single business document in the ANSI X12 standard is a transaction set, which will be described fully in a moment. The EDI equivalent of the traditional envelope is a data structure that carries sufficient data to ensure that the transaction sets enclosed within are sent to the appropriate recipient. This part of the ANSI X12 data structure is not discussed further here

b. Transaction Sets

A transaction set is a single business document, such as a purchase order or an invoice. In ANSI X12 each transaction set is assigned a single three digit identifier and these numbers are often used in place of the transaction set descriptions. A purchase order is thus referred to as an "850" and an invoice is referred to as an "810".
Transaction sets are collections of sequenced records, which are termed segments in the terminology of ANSI X12. These segments are further grouped in defined areas of the document, referred to in ANSI X12 as tables, which generally include a header, detail, and summary area.

Table 1 - Header Area
Table 2 - Detail Area
Table 3 - Summary Area

Segments that can be used in each area are defined in the ANSI X12 manual. Some segments can be used in more than one area, and the importance of this fact will be discussed later.

The transaction set details in the ANSI X12 manual show which segments may be used in the transaction set, and required sequence of the segments. For instance, here are the first few segments of the purchase order transaction set :-

ID	Description	
	Usage	Max Use
ST	Transaction Set Header	
	M	1
BEG	Beginning Segment for Purchase Order	
	M	1
NTE	Note/Special Instruction	
	F	100
CUR	Currency	
	O	1
REF	Reference Numbers	
	O	12
PER	Administrative Communications Contact	
	O	3

etc

Transaction sets start with the Transaction Set Header (ST) segment, and end with the Transaction Set Trailer (SE) segment. These two segments are the innermost level of the three levels of "electronic envelopes" within the ANSI X12 data structure.

Each segment in the transaction set is defined as mandatory (M) or optional (O), and can repeat many times at its particular location within the transaction, if its maximum use is defined at a value greater than one. The exception to this rule is the Note (NTE) segment, which can be defined as a floating (F) segment. This means it can appear anywhere in the transaction !. Understandably, its usage is beginning to be questioned, and new transactions generally specify the location of Note segments with mandatory or optional designations.

c. Loops
Some of the segments in a transaction set can also be grouped together in <u>loops</u>. These segment groupings are named after the first segment in the loop, and the segment group may repeat many times at that location in the transaction set.

For instance, here is an extract from the purchase order transaction set table :-

ID	Description	Usage	Max Use	Loop
ST	Transaction Set Hdr	M	1	
BEG	Beginning Segment	M	1	
NTE	Note/Special Inst.	F	100	
CUR	Currency	O	1	
REF	Reference Numbers	O	12	
PER	Admin. Contact	O	3	
etc				
MAN	Marks and Numbers	O	10	
N1	Name	O	1	
N2	Additional Name Info	O	2	
N3	Address Information	O	2	
N4	Geographic Location	O	1	N1/200
REF	Reference Numbers	O	12	
etc				
PKG	Marking, Pckg. etc	O	25	

This example shows that the name and address loop beginning with the N1 segment can occur up to 200 times in the purchase order transaction. Examples of its use would be for bill to and ship to addresses. It should also be noted that the N1 loop is optional, but if it is it be used, the N1 segment is mandatory. This example also shows the importance of the segment sequence, as the only distinguishing feature between the two occurrences of the REF segment is that the second occurrence will follow an N1 segment, if it is used.

d. Segments

Segments are variable length records which consist of a collection of logically related variable length fields, or data elements. Each segment is identified by a record identifier called a segment identifier, and the record is terminated by a mutually-defined record terminator character called the segment terminator. Finally, each data element in the segment is separated from the neighboring data elements by a mutually-defined field separator character called a data element separator.

The following example shows a purchase order line item represented in the ANSI X12 standard format :-

```
           PO1*2*12*EA*.47**CB*B-974*VC*06-121-47~
           ^       ^        ^  ^                 ^
           |       |        |  |                 |
segment identifier |        |  |         seg terminator
                   |        |  |
     data element separator |  qualified data element
                            |
                            qualifier
```

This segment represents the second line item of a purchase order, in which 12 products are being purchased at a unit price of $0.47. The customer's catalog number is B-974, and the vendor's catalog number is 06-121-47.

e. Qualified Data Elements
The segment shown above also illustrates the concept of qualified data elements. A qualified data element is one whose contents are only fully known when a preceding data element is examined. For instance, in this example, we know that B-974 is the customer's catalog number because the preceding data element is CB, which represents the buyer's catalog number.

f. Qualified Loops
Loops can also be qualified according to a data element in the first segment of the loop. The example below shows several segments in an N1 loop for a ship to address :-

```
N1*ST*THE CORNER STORE*92*1001~
N3*601 FIRST STREET~
N4*CROSSROADS*MI*48106~
```

The only indicator that all these segments contain data relating to a ship to address is the first data element in the N1 segment which contains the code ST.

The preceding description of the data structure used for ANSI X12 transaction sets is by no means exhaustive, but should serve to allow the reader to understand the enormous flexibility that it affords both the designers of transaction sets and their end users (your trading partners !).

3. Format of ANSI X12 Purchase Order

a. Paper representation of purchase order

PO No	80003	PO Date	01/13/92
Requested Ship Date	01/27/92	Cancel Date	01/31/92

Bill To	Ship To
Acme Distributing Company P.O. Box 33327 Anytown, NJ 44509	The Corner Store (#1001) 601 First Street Crossroads, MI 48106
If any questions contact	Jon Smith at 313 888 4356
Seller	Buyers Number for Seller 42168
Smith Corporation 900 Easy Street Big City, NJ 15455	
Terms of Payment	15 days from receipt of invoice
Terms of Shipment	Prepaid by seller

Item	Buyer Part	Seller Part Part	Qty	UOM	Price	Desc
1	A-493	01-340-02	3	Case	12.75	C/Sponges
2	B-974	06-121-47	12	Each	0.47	P/Pails

b. ANSI X12 representation of purchase order

```
ST*850*2650                                    <--- Header Area
BEG*00*SA*80003***920113~                           |       (Table 1)
FOB*PP~                                             |
ITD*05*5*****15~                                    |
SHH*SC*010*920127~                                  |
SHH*ZZ*001*920131~                                  |
N1*BT*ACME DISTRIBUTING COMPANY~                    |
N3*P.O. BOX 33327~                                  |
N4*ANYTOWN*NJ*44509~                                |
N1*ST*THE CORNER STORE*92*1001~                     |
N3*601 FIRST STREET~                                |
N4*CROSSROADS*MI*48106~                             |
PER*BD*JOHN SMITH*TE*3138884356~                    |
N1*SE**92*42168~                               <---
PO1*1*4*CA*12.75**CB*A-493*VC*01-340-02~ <-- Detail Area
PID*F****CELLULOSE SPONGES~                        |  (Table 2)
PO1*2*12*EA*.47**CB*B-974*VC*06-121-47~            |
PID*F****PLASTIC PAILS~                          <--
CTT*2*16~                                       <--- Summary Area
SE*20*2650~                                     <--- (Table 3)
```

4. Where can variations in format occur ?

The example purchase order shown above displays one method of constructing an ANSI X12 purchase order (850) transaction set from the available data. We will now examine some of the alternative ways to construct an 850 transaction set with this same data, whilst remaining within the confines of the ANSI X12 standard.

a. Segment or loop in different tables
Every segment and loop following the BEG segment in the header area (Table 1) of the purchase order (850) transaction set shown above can be placed either at this level or in the detail area (Table 2). Where both appear, the detail level data overrides the header level data. The implications of this variation for receiving EDI orders and loading them into a sales order processing system are considerable.
Let us look at the type of data contained in the header area which can also be placed at the detail level. In this example, there are shipment scheduling dates and addresses (bill to and ship to) which could either be placed at the header level or in each line item in the detail area. This would be fine if all the scheduling dates and addresses were the same for each line item, but what if they differ?. This is a major area of concern for application integration which will discuss further.

b. Different segments used for same purpose
The SHH (General Schedule) segments shown above, which contain the requested ship date and the order cancellation date, could be replaced at almost the same place in the transaction set with the following DTM (Date/Time) segments :-

```
DTM*010*920127~
DTM*001*920131~
```

c. Data placed at multiple positions in same segment
In the purchase order line item shown above there are two positions in which the part number can be placed. In fact, the standard allows for a total of 10 different product identifiers on the purchase order line item. The product identifiers used are not tied to any specific positions in the segment, so that the seller's part number could appear in any of the ten positions. Both the type of product

identifiers included and their positions within this segment are usually at the discretion of the customer. Customers may include any combination of product identifiers and place them in any of the ten positions available.

d. Line item quantity broken down to multiple ship to locations
The example above shows two products which need to be delivered to ship to location 1001. What if the buyer wanted half the products delivered to ship to location 1001 and half to 1002 ?. Generally, this would require that two purchase orders were raised, one for each location. However, some purchasing and sales order processing systems allow for multiple ship to locations for each line item, so the 850 transaction set supports this requirement using the SDQ (Destination Quantity) segment.

The detail area shown above could be reconstructed as follows to include this requirement :-

```
PO1*1*4*CA*12.75**CB*A-493*VC*01-340-02~
PID*F****CELLULOSE SPONGES~
SDQ*CA*92*1001*2*1002*2~
PO1*2*12*EA*.47**CB*B-974*VC*06-121-47~
PID*F****PLASTIC PAILS~
SDQ*EA*92*1001*6*1002*6~
```

In this example, the SDQ segment looks rather tame, but it contains up to 10 ship to locations and can occur up to 500 times for each line item. This means that a large retail customer like K Mart or Wal Mart could break down a single line item quantity within a purchase order into individual shipment quantities for up to 5000 ship to locations !.

These examples of the different ways in which the 850 transaction set can be constructed demonstrate that it is highly unlikely that all of your customers will submit EDI orders to you in a single format. The particular format that they adopt is likely to mainly influenced by the guidelines which may exist in their industry for the formatting of EDI purchase orders. Another factor which is likely to influence the purchase order format that a particular customer adopts is the procurement system used to generate the orders. We will now consider which of these variations are handled by EDI software packages, and which need to be considered when building interface programs to load the output from EDI software into an application system.

5. What variations do EDI software packages handle ?

The first generation of EDI software packages were developed to address the need to translate EDI transaction sets into files with fixed length fields within fixed length records. The term translation software was adopted to describe these packages, and in the early days they merely transformed each of the data elements received in each segment into fixed length fields. This produced files with all of the received segments transformed into directly equivalent fixed length records with fixed length fields.

If documents represented in an EDI standard format were defined in a completely unambiguous way, translation would be the only function which was necessary to interface EDI data to business applications. However, as we have seen above, within a particular transaction set, such as a purchase order, there are many options for both the placement of particular data within the transaction set and the type of data which is included.

Variations in the completion of transactions in an EDI standard format need to be addressed in either EDI software or the programs which interface EDI data to the in-house application systems. As it is not desirable to have to modify application interface programs for every new trading partner that is added, EDI software companies developed mapping software to address these problems. The main object of mapping is to transform any format used by trading partners into a neutral format which is static and which can be used to interface to in-house systems. Mapping software was originally developed to remap the output from translation into a flat file format which could be easily interfaced to existing systems. More recently, the mapping component of EDI software has been integrated with the translation process to allow EDI data to be mapped directly into the desired flat file format.

The design of the flat file to be interfaced between the EDI software package and the application system is critical, and needs to be based much more on the data requirements of the in-house application than the data typically sent or received in EDI transactions. Any variation in format from each trading partner should ideally be handled by the EDI software package. EDI software should insulate the in-house application from these variations and allow the development of application interface programs which operate with static interface file formats.

However, in practice, there are certain variations that EDI software handles well, and certain variations that it is more appropriate to handle in the application interface programs. EDI software packages strengths generally revolve around the transformation of the data received in the EDI transaction set into a format which can be readily accepted by the application interface program. This transformation can include :-

a. Mapping data selectively by Trading Partner
For instance, for customer A this field in the application interface file is found in the SHH segment of the EDI order with a qualifier of 001, but for customer B it is retrieved from a DTM segment with the same qualifier.

b. Cross reference tables by Trading Partner
For example, not all customers are benevolent enough to transmit your own part number as part of the line item detail, so cross reference tables need to be used to convert their part number to your own.

c. Variable Substitution
Placing today's date or time into a particular field in the interface file, for example.

d. String manipulation
Concatenating data elements together or extracting substrings, for example.

e. Math
Calculating line item extensions or performing counts, for instance.

The areas of variation that are better handled by application interface programs are those which involve decisions which need to be made about documents whose underlying data structure is radically different from the in-house application. For instance, if the in-house sales order processing system can only handle one ship to location per order, any order which has multiple ship to locations per line item will need to be pre-processed to break up the order into component orders by ship to location, which can then be loaded into the application. One approach to handling this situation would be as follows. The application interface file produced by the

EDI software would include fields for ship to location information at both the header and detail level. The application interface program would then include logic to perform certain functions if ship to location information appeared at (a) the header level, (b) the detail level, or (c) both header and detail levels. Approaching the problem from this angle allows a single application interface file to be established for incoming orders which can accommodate both single ship to location orders and multiple ship to location orders.

6. Initiatives to reduce the variations in format

The discussion so far has considered what variations occur in the EDI data that trading partners can transmit and how to handle those variations using EDI software and application interface programs. However, wouldn't it be better to try to solve the problem at its source rather than find ways to cope with the problem after the event has happened ?. If companies could agree within a particular industry, or preferably across all industries, a common approach to the placement of particular data within existing EDI transactions, the task of interfacing EDI data to applications would be much more straightforward. To some extent, the implementation guidelines for EDI transactions issued by certain industry action groups like the Automotive Industry Action Group (AIAG) already address this issue. However, these guidelines usually still allow significant variations to occur, as they were generally drawn up by merging the formats already in use by major trading partners within the industry.

The AIAG has recognized this problem and has decided to bring together the major automotive companies and their major suppliers to examine two EDI transactions in common usage in that industry. Their objective is to reduce the variations imposed by particular customers on these transactions to an absolute minimum, to enable the automotive suppliers to reap the benefits of EDI more easily. In the initial 2 day AIAG meeting to begin this initiative, the participants managed only to define the scope of the problem by documenting where variations between the different implementations of these transaction sets occurred. For transactions which are already widely used in the automotive industry, the task of consolidating the various company specific formats is likely to take some time. However, one of the unexpected benefits of bringing together the automotive companies and suppliers in this way, has been the fact that when automotive companies are considering implementing new transactions they examine their competitors implementations and try to use their placement of data wherever possible.

Another initiative that is under way within ANSI X12 is approaching the problem of implementation variations from a cross industry standpoint. This group is currently working on a project to enable both the standards developers and industry action groups to examine the usage of segments and data elements within a particular transaction set across all published implementation guidelines. The output from the project will probably be a PC-based software product which will (i) allow new industry action groups to define implementation guidelines for EDI transactions which require the placement of data within in the transaction according to the most common usage across all industries, and (ii) allow standards developers to re-examine the usage of segments and data elements within existing transaction sets and, hopefully, to eliminate those which are not currently in use.

Both these initiatives are likely to be of more benefit for new transactions than for existing ones, primarily because companies participating in EDI today have large investments in application integration designed to support their own specific implementations of the standards.

7. Summary

The most common misconception about EDI is that business documents are replaced by transaction file formats that are unambiguous. I hope that the preceding overview of the variations in format of the ANSI X12 purchase order transaction serves to dispel that misconception. The purpose of describing these variations is not to discourage you from participating in the EDI process with your trading partners, but rather to make you aware of some of the issues that are likely to arise when your EDI implementation program extends beyond the first trading partner. EDI software can certainly help to insulate your in-house applications from these variations. It has evolved over the past ten years to address the problems encountered by the early implementors, and it will continue to evolve to ease integration of EDI data with business applications by using techniques which make the management of multiple formats of incoming (and outgoing) transactions much easier.

PAPER NO.: 3112

TITLE: Integrated EDI: Is It Possible?

AUTHOR: Rod E. Keiser
Computer Management Solutions
7208 South Tucson Way Suite 175
Englewood, CO 80112
(303) 790-0123

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

HOW TO LASER AWAY PREPRINTED FORMS

Jason Kent and Roger Lawson
Proactive Systems Inc.
4 Main Street
Los Altos
CA 94022
(Tel: 415-949-9100)

ABSTRACT

This talk will explain the benefits of designing and printing forms using laser printers instead of conventional preprinted stationery.

The benefits to be discussed will include the savings in design time, savings in set-up time, less operator intervention, lower print costs, more timely printing (single sheets possible instead of batch printing), higher quality printing, the ability to print locally to the user, the general advantages of laser printers versus impact printers and many other aspects.

The added flexibility of being able to vary forms from page to page, or to select forms under program control will be covered.

How to implement such systems without changing existing applications will be discussed.

To the junk heap

I believe the conventional computer line printer will be on the junk heap in a couple of years time. The chain, band, and dot matrix printers are inflexible, noisy, cumbersome to operate and less reliable than laser printers such as the HP LaserJet Series III or HP 5000.

Laser Printing Advantages

With LaserJets you get total flexibility in layout and typeface selection. Why stick to the fixed pitch typewriter-like look of a conventional computer printer when you can make your output look professionally typeset for no extra cost?

With LaserJets you can place graphics (e.g. bar/line charts, logos, drawings, etc.) among your text or data to enhance its quality and impact. And every copy comes out clean, crisp and clear.

LaserJets are silent, use standard sheet paper and changing paper is easy. Compare the ease of use of a LaserJet Series III and a common conventional HP printer such as the 2932 to see what I mean.

You can use a small font and print more information on less paper (and, on some models, print on both sides of the paper) so as to save paper and hence money. In fact the running costs (e.g. consumables, paper, maintenance) can be less than conventional computer printers.

Incidentally you can of course use an ordinary LaserJet as a spooled device on an HP3000 or HP9000 and the print speed is faster than many people realise. For example at 8 pages per minute (p.p.m.) a Series III LaserJet can be equivalent to a 400 lines per minute (l.p.m.) conventional printer. The HP 5000 models now go up to 135 p.p.m. and you can in fact print two US Letter side pages together on one page because of the 17 inch throat so effectively doubling the pages per minute.

The Software Problem

However most LaserJet users are not taking full advantage of these printers because of the difficulty of using their sophisticated capabilities. Unless you have some special software, the only way to format output is with long, complex escape sequences - for other than simple applications this is simply not practical or economic in labour.

What do we need in software to drive them from the standpoint of the commercial minicomputer user? My list would be:

■ Flexible font control and typesetting capabilities, e.g. the ability to justify text in a proportional type face, do multi-column output, do automatic headers and footers, etc.

■ Graphics functions such as drawing lines and boxes plus the ability to draw commonly used figures such as line, bar and pie charts. This should also extend to the easy design and production of "forms", i.e. grid based artwork (see Figure A for an example).

■ The ability to control or drive this software from existing programs written in COBOL, 4GLs, report writers, etc.

■ The ability to intercept printer output from existing programs so that source code changes are not required.

■ Software that is shareable by multiple users, that runs efficiently (due to the high volumes of data to be processed) and that can be set up to run in batch jobs without operator intervention.

Note that Desk Top Publishing (DTP) packages as used on PCs do not meet several of the above criteria so we really need a different "animal" for the typical HP3000 or HP9000 user.

Electronic Forms

A large volume of the paperwork in most organisations is a "form" which is basically a grid with some typeset text. The grid is used to logically organise the material and to enable a "denser" layout than would otherwise be acceptable.

If computer printing is required on a form then "preprinted stationery" is obtained where the form is printed on continuous paper. Such paper has to be specially mounted on a printer and this makes it impractical to use for applications where one-off sheets need to be printed on an ad-hoc basis.

Both types of form are expensive to design and print, they are expensive to distribute and hold in stock, and a lot of them get scrapped when the form needs to be changed! For example I have seen an industry survey that showed that 40% of all pre-printed forms are scrapped due to obsolescence.

Also the form is fixed in layout - you can't change it from page to page. A simple example of this requirement is an invoice on which a total line is needed - however if an invoice for one particular customer extends over several pages you only want the total printed on the last page.

All of these problems can be solved if you hold the form as an "electronic" equivalent and only print it when required. If the form needs amendment (e.g. maybe your company address has changed) this is a trivial exercise.

You can print a "blank" form; or you can set it up so that a user can fill out the form on a terminal and print it when finished (or you can get the form filled automatically by the computer from its data base before printing).

LaserJet Macro Capability

One reason why forms can be efficiently produced on a LaserJet is because they can be downloaded and held in the LaserJet memory (more than one can be held depending on the amount of memory). Therefore the form and data can be merged in the LaserJet rather than on the host system.

By holding the "form" as electronic software it is also easy to pass it around the organisation and to enforce standard usage. Figure A is an example of such an electronic form designed and printed on an HP3000.

Improving Reports and Textual Forms

It is easy to use simple graphic design tricks such as lines and boxes to improve the appearance of computer output reports enormously. Also putting it into a decent typeface makes it more legible.

Look at figures B and C for example. The one printed on the laser printer didn't require much more effort to design and specify but it certainly looks a lot better - also the 2-column layout saves paper. This type of output can be conceptually handled as a "form" also, but note how the data wraps around in a two column layout.

Many forms are also primarily textual - see Figure D for such an example. In this case dynamic typesetting capabilities are more important than the ability to draw grid artwork.

Dynamic Forms

By printing forms on a LaserJet using appropriate software, you can actually vary the content of the form from page to page. A couple of examples from our own customers are as follows:

■ An application form for insurance is varied depending on the basic information already known about the applicant (e.g. whether he is an existing policy holder or not) - some parts of the form that are therefore irrelevant are omitted.

■ A company that distributes office consumables, supplies each of its customers with an order form which they can distribute internally. Only the items that the customer has agreed are relevant to his organisation are listed on the form (these items are listed in the suppliers data base and the agreed purchase terms also specified). As the supplier has many customers, there are many different versions of the form which are produced automatically. If an additional item is to be added to the "orderable" list then the program that prints the order form automatically adapts it.

Software Capabilities

At this point it is worth looking at the requirements for laser printing software. One aspect to consider is the typesetting capabilities it provides. For example, the following text is printed in a fixed pitch Courier font which is a typical typeface used by typewriters or conventional computer printers.

```
This text is an example of a paragraph that
is printed in different typefaces to show
the varying capabilities of different fonts
and the ability of typesetting software to
format the text in a better manner.
```

However on a LaserJet you can use professional proportional typefaces such as Times Roman or Helvetica. The example below is the same text in Helvetica 12 point (the size of type is measured in points there are 72 points in an inch). It is called a proportional font because the amount of space ocupied by each character in the alphabet varies. Also to make it look even better we can use kerning (varying the space between characters depending on the character pair) and justification (in this case to both margins which means the space between words is varied also). The sample text then appears as:

This text is an example of a paragraph that is printed in different typefaces to show the varying capabilities of different fonts and the ability of typesetting software to format the text in a better manner.

Note also that the type size and form can be varied within a line thus adding further complexity. Obviously with this dynamic formatting process you also need automatic hyphenation for long words as the person entering the text cannot easily predict where hyphenation will be required.

Fonts on the LaserJet can come from three sources: in-built, cartridge or downloaded soft fonts. The last is the most flexible although it obviously takes some time to download. On LaserJet III models there are proportional fonts built-in. In practice the standard LaserJet memory is sufficient to hold a number of fonts in commonly used sizes so a single download is usually sufficient. Incidentally some software packages include the commonly used proportional fonts.

To cover all the above, and more, in a flexible and sophisticated manner, requires good software. The quality of software packages aimed at LaserJet formatting varies greatly in the quality and sophistication of facilities in this area so you should look carefully at that - even a simple form tends to contain a lot of text so such capabilities are important.

Graphics, Logos and Bar Coding

Graphics is still one of the Cinderellas of the commercial computing world. With LaserJet printers it is perfectly practical to incorporate graphics such as bar charts or logos into forms output.

There are two ways of drawing graphics on a LaserJet. The first way is to use raster graphics where each dot to be printed is represented by a binary value. Building up a picture in this way requires transmitting a large amount of data to the printer (bearing in mind that a LaserJet prints 300 dots per inch) so using this method for routine production graphics is not a good idea, for example it could take several minutes to print one page. However it is fine for smallish logos, particularly if it repeated the same on every page, in which case it can be held in the printer memory.

The second method is to use the PCL command language built into the printer. This effectively enables you to draw lines. It cannot achieve as complex a drawing as raster but for bar, line and pie graphs it is fine.

A particular requirement when printing forms is often to include a corporate logo. This can often be drawn with PCL based graphics. Alternatively it can be scanned as a raster graphics using an HP ScanJet attached to a PC and uploaded to the HP3000/HP9000. You can also often turn logos into a font for more convienient use.

Another common application for PCL graphics is bar code printing. Many companies are now implementing bar coding on product labels and turn-around documents, so as to save data entry time. Although it is possible to print bar codes on dot matrix or other impact printers, they are very easily produced on a laser printer and are of very high quality in that case. You don't even need to use a font cartridge if you have good software, as bar codes can be very quickly drawn on the fly using PCL commands.

What Does the HP3000 or HP9000 User Need?

If you look at software to support LaserJet printers (ie. which can provide the functions described above) it may at first appear that a PC based product would provide a solution. However the needs of a minicomputer user differ considerably from that of a PC user and the software must differ accordingly. For example the typical HP3000 or HP9000 user needs the following:

■ The ability to handle large data volumes with good performance is a priority (a good question to ask is whether the software tracks forms and fonts that are downloaded to printers so as to minimise redownloads).

■ The support of multiple spooled devices, i.e. possibly several LaserJets of different models which have different capabilities. For example, can you configure different printer memories, can you tell the software which printers have in-built scalable fonts, does the software know which printers support HP-GL which is much more efficient for drawing curves than PCL? Also does the software cover the need to distribute reports across different printers automatically?

■ The need to support lots of terminal devices (of which some will be PCs but some will also be "dumb" terminals). Does the software support both DOS and MS-WINDOWS PC form design?

■ A requirement to interface to (and be useable by) software written in COBOL, "C", 4GLs (such as Powerhouse, Speedware), or report writers such as QUERY, QUIZ, Q-GEN, BRW etc.

■ The ability to format output from existing application packages for which the source code may not be available, eg. can it pick up and apply electronic forms to existing output without even changing the application job control?

■ System management functions should be available to configure output devices, maintain a catalog of forms in use, and otherwise manage the shared resource.

■ A cross-platform solution is required by many users, i.e. does it run on HP3000, HP9000 (and other UNIX, or open systems based platforms) and on PCs with shareable forms?

■ It is worth examining whether the software can just handle grid type forms, or whether it can also handle textual type forms, support report enhancement, and includes graphic capabilities.

The Advantages of Electronic Forms

To summarise, these are the advantages of replacing preprinted forms by electronic forms:

1. You can use plain paper stock, at lower cost, than using preprinted forms.

2. You can get a new form designed and in use in minutes, as against the days or weeks it takes to get a preprinted form produced.

3. You don't have to hold a large inventory of preprinted forms, which occupies expensive office space, or worry about re-ordering them.

4. Modifying an existing form can be done in seconds, and you never need to scrap existing paper stocks when they become obsolete.

5. Staff are not required to change forms on a printer. It can all be done under program control.

6. Laser printing produces a better quality result than impact printers. You can include data-driven graphics and bar codes also.

7. You can easily distribute standard forms around an organisation in electronic form (and even transmit them over a phone line).

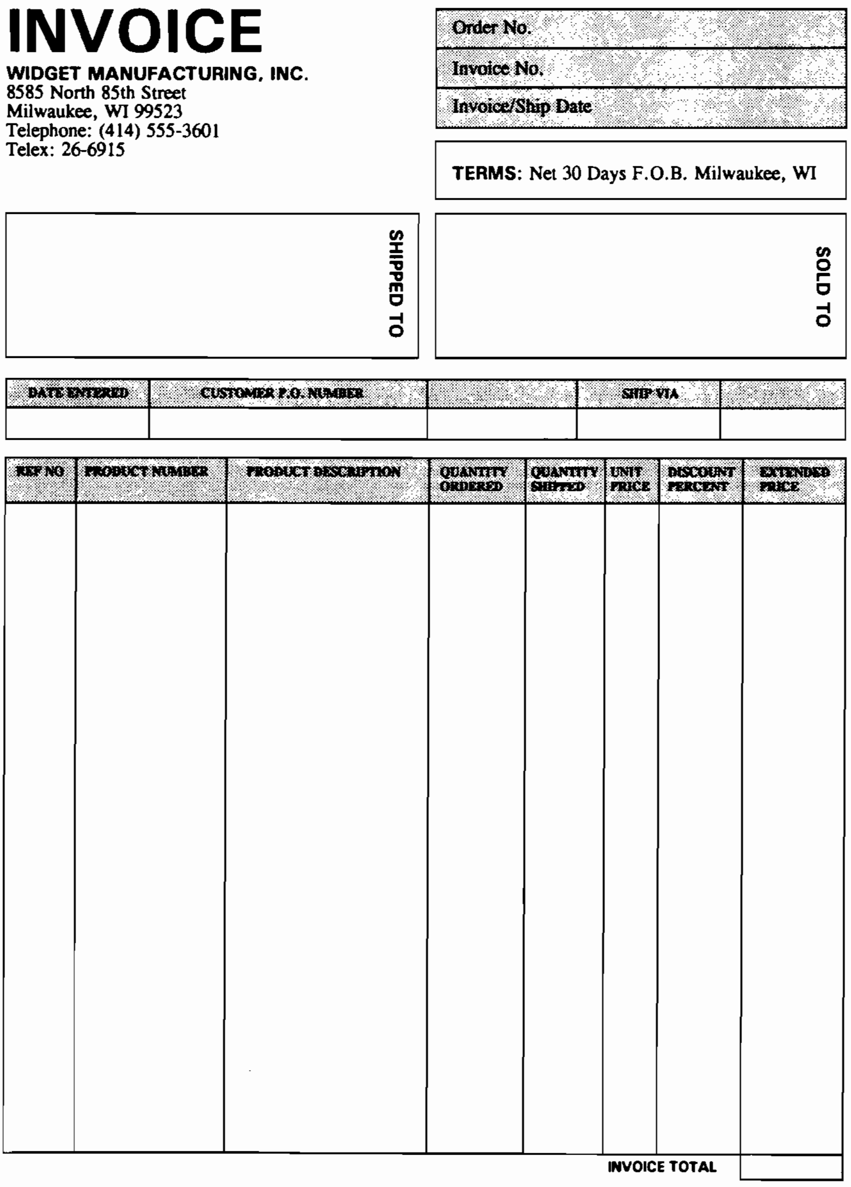
8. You can print different forms on every output page, and there is no set-up overhead so you no longer need to print things in batch mode.

9. You can print forms that adapt to the data in a dynamic way, ie. you have absolute program control of the output.

10. You can use quiet, low cost, laser printers to provide a distributed printing solution so that output prints in the users own office, rather than in some centralised facility.

11. There is no more need to burst and decollate output, or to physically distribute it.

FIGURE A



INVOICE

WIDGET MANUFACTURING, INC.
8585 North 85th Street
Milwaukee, WI 99523
Telephone: (414) 555-3601
Telex: 26-6915

Order No.

Invoice No.

Invoice/Ship Date

TERMS: Net 30 Days F.O.B. Milwaukee, WI

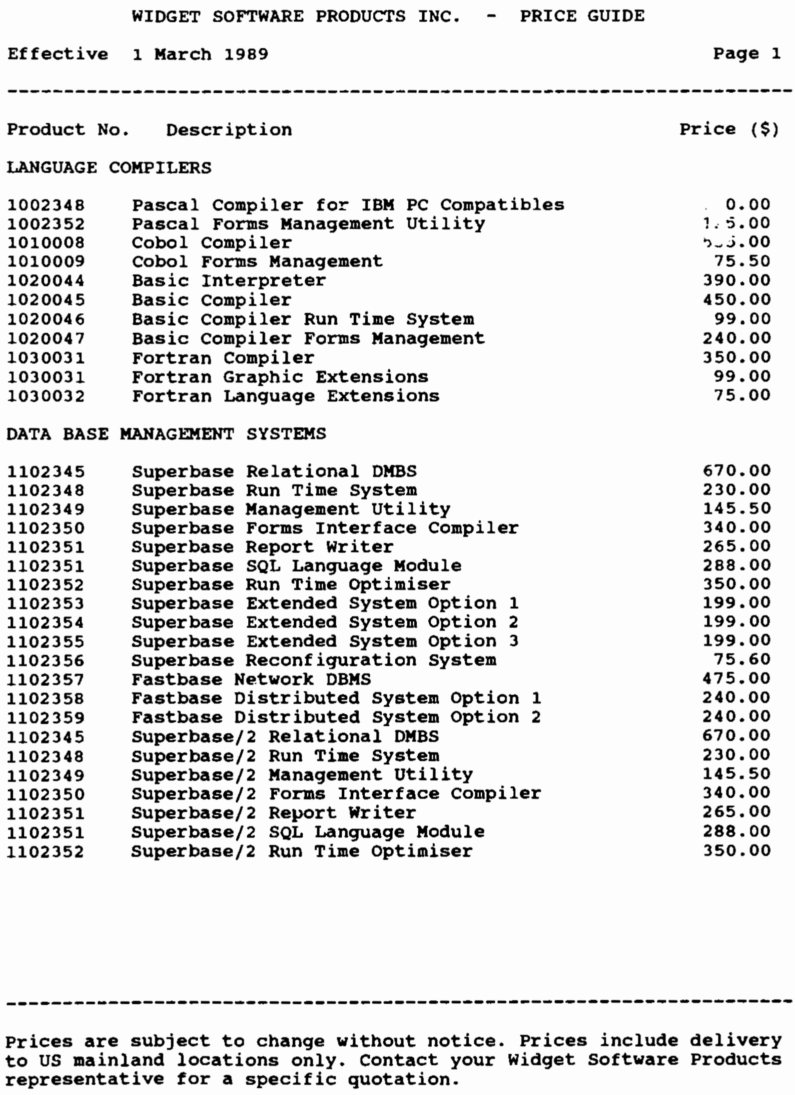
SHIPPED TO

SOLD TO

DATE ENTERED	CUSTOMER P.O. NUMBER		SHIP VIA	

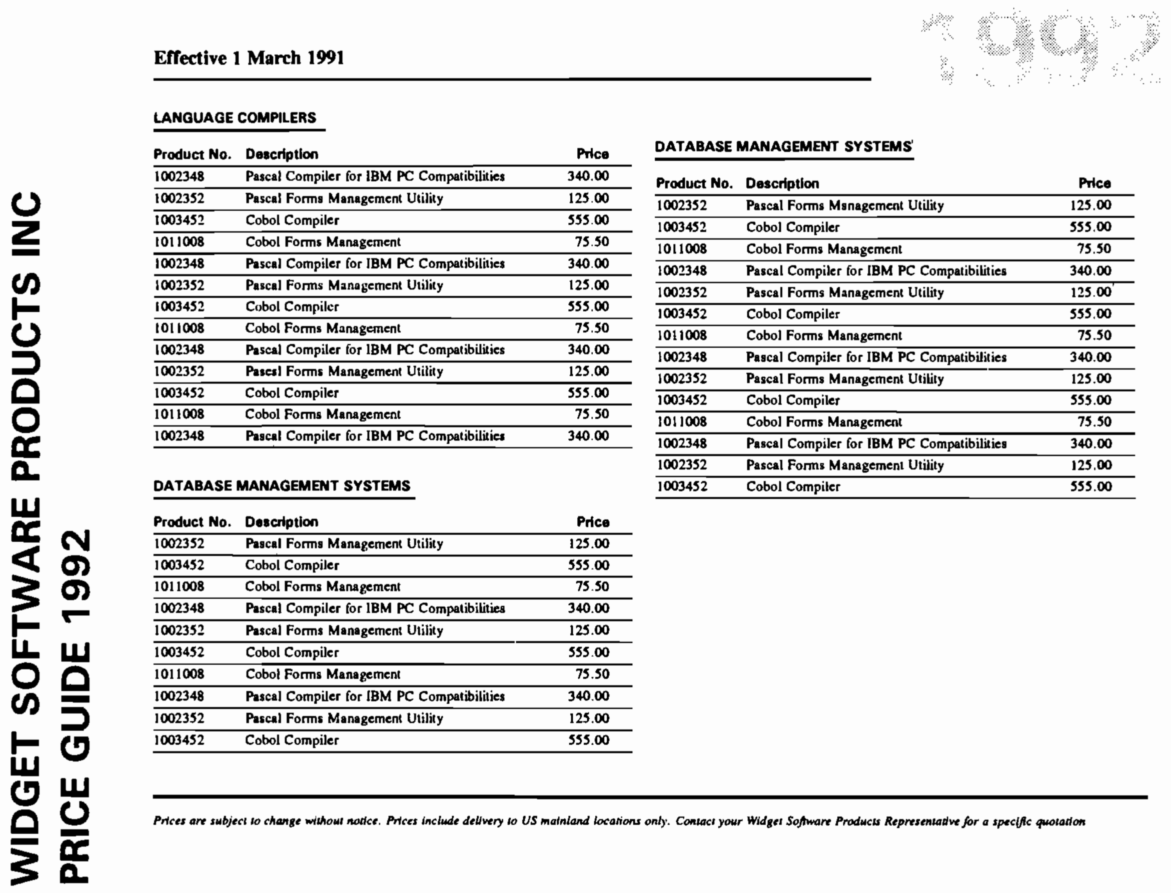
REF NO	PRODUCT NUMBER	PRODUCT DESCRIPTION	QUANTITY ORDERED	QUANTITY SHIPPED	UNIT PRICE	DISCOUNT PERCENT	EXTENDED PRICE
						INVOICE TOTAL	

FIGURE B



```
            WIDGET SOFTWARE PRODUCTS INC.  -  PRICE GUIDE

Effective  1 March 1989                                      Page 1
---------------------------------------------------------------------

Product No.   Description                                 Price ($)

LANGUAGE COMPILERS

1002348     Pascal Compiler for IBM PC Compatibles             0.00
1002352     Pascal Forms Management Utility           1[illegible]5.00
1010008     Cobol Compiler                            5[illegible]5.00
1010009     Cobol Forms Management                            75.50
1020044     Basic Interpreter                                390.00
1020045     Basic Compiler                                   450.00
1020046     Basic Compiler Run Time System                    99.00
1020047     Basic Compiler Forms Management                  240.00
1030031     Fortran Compiler                                 350.00
1030031     Fortran Graphic Extensions                        99.00
1030032     Fortran Language Extensions                       75.00

DATA BASE MANAGEMENT SYSTEMS

1102345     Superbase Relational DMBS                        670.00
1102348     Superbase Run Time System                        230.00
1102349     Superbase Management Utility                     145.50
1102350     Superbase Forms Interface Compiler               340.00
1102351     Superbase Report Writer                          265.00
1102351     Superbase SQL Language Module                    288.00
1102352     Superbase Run Time Optimiser                     350.00
1102353     Superbase Extended System Option 1               199.00
1102354     Superbase Extended System Option 2               199.00
1102355     Superbase Extended System Option 3               199.00
1102356     Superbase Reconfiguration System                  75.60
1102357     Fastbase Network DBMS                            475.00
1102358     Fastbase Distributed System Option 1             240.00
1102359     Fastbase Distributed System Option 2             240.00
1102345     Superbase/2 Relational DMBS                      670.00
1102348     Superbase/2 Run Time System                      230.00
1102349     Superbase/2 Management Utility                   145.50
1102350     Superbase/2 Forms Interface Compiler             340.00
1102351     Superbase/2 Report Writer                        265.00
1102351     Superbase/2 SQL Language Module                  288.00
1102352     Superbase/2 Run Time Optimiser                   350.00



---------------------------------------------------------------------

Prices are subject to change without notice. Prices include delivery
to US mainland locations only. Contact your Widget Software Products
representative for a specific quotation.
```

FIGURE C



WIDGET SOFTWARE PRODUCTS INC

PRICE GUIDE 1992

1992

Effective 1 March 1991

LANGUAGE COMPILERS

Product No.	Description	Price
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00

DATABASE MANAGEMENT SYSTEMS

Product No.	Description	Price
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00

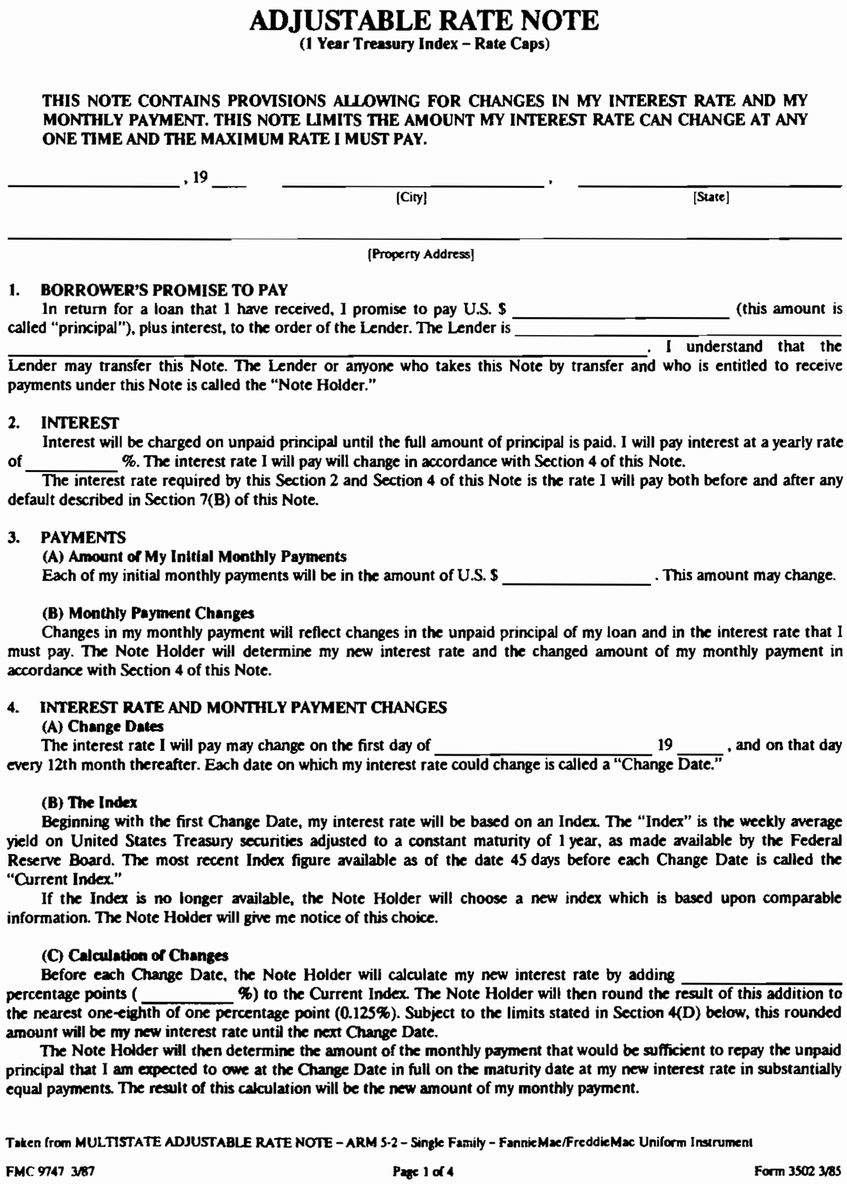
DATABASE MANAGEMENT SYSTEMS

Product No.	Description	Price
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1011008	Cobol Forms Management	75.50
1002348	Pascal Compiler for IBM PC Compatibilities	340.00
1002352	Pascal Forms Management Utility	125.00
1003452	Cobol Compiler	555.00

Prices are subject to change without notice. Prices include delivery to US mainland locations only. Contact your Widget Software Products Representative for a specific quotation

3113-11
How to Laser Away Preprinted Forms

FIGURE D



ADJUSTABLE RATE NOTE

(1 Year Treasury Index – Rate Caps)

THIS NOTE CONTAINS PROVISIONS ALLOWING FOR CHANGES IN MY INTEREST RATE AND MY MONTHLY PAYMENT. THIS NOTE LIMITS THE AMOUNT MY INTEREST RATE CAN CHANGE AT ANY ONE TIME AND THE MAXIMUM RATE I MUST PAY.

____________________, 19____ ____________________________, ______________________________

[City] [State]

____

[Property Address]

1. BORROWER'S PROMISE TO PAY

In return for a loan that I have received, I promise to pay U.S. $ ____________________ (this amount is called "principal"), plus interest, to the order of the Lender. The Lender is ______________________________ ____. I understand that the Lender may transfer this Note. The Lender or anyone who takes this Note by transfer and who is entitled to receive payments under this Note is called the "Note Holder."

2. INTEREST

Interest will be charged on unpaid principal until the full amount of principal is paid. I will pay interest at a yearly rate of __________ %. The interest rate I will pay will change in accordance with Section 4 of this Note.

The interest rate required by this Section 2 and Section 4 of this Note is the rate I will pay both before and after any default described in Section 7(B) of this Note.

3. PAYMENTS

(A) Amount of My Initial Monthly Payments

Each of my initial monthly payments will be in the amount of U.S. $ ________________. This amount may change.

(B) Monthly Payment Changes

Changes in my monthly payment will reflect changes in the unpaid principal of my loan and in the interest rate that I must pay. The Note Holder will determine my new interest rate and the changed amount of my monthly payment in accordance with Section 4 of this Note.

4. INTEREST RATE AND MONTHLY PAYMENT CHANGES

(A) Change Dates

The interest rate I will pay may change on the first day of ________________________ 19_____, and on that day every 12th month thereafter. Each date on which my interest rate could change is called a "Change Date."

(B) The Index

Beginning with the first Change Date, my interest rate will be based on an Index. The "Index" is the weekly average yield on United States Treasury securities adjusted to a constant maturity of 1 year, as made available by the Federal Reserve Board. The most recent Index figure available as of the date 45 days before each Change Date is called the "Current Index."

If the Index is no longer available, the Note Holder will choose a new index which is based upon comparable information. The Note Holder will give me notice of this choice.

(C) Calculation of Changes

Before each Change Date, the Note Holder will calculate my new interest rate by adding ________________ percentage points (__________%) to the Current Index. The Note Holder will then round the result of this addition to the nearest one-eighth of one percentage point (0.125%). Subject to the limits stated in Section 4(D) below, this rounded amount will be my new interest rate until the next Change Date.

The Note Holder will then determine the amount of the monthly payment that would be sufficient to repay the unpaid principal that I am expected to owe at the Change Date in full on the maturity date at my new interest rate in substantially equal payments. The result of this calculation will be the new amount of my monthly payment.

Taken from MULTISTATE ADJUSTABLE RATE NOTE – ARM 5-2 – Single Family – FannieMae/FreddieMac Uniform Instrument

FMC 9747 3/87 Page 1 of 4 Form 3502 3/85

INCORPORATING DYNAMIC ELECTRONIC FORMS INTO HP 3000 APPLICATIONS

Ross G. Hopmans
Brant Technologies Inc.
51 Tannery Street
Mississauga, Ontario
Canada L5M 1V3
(416) 858-0153

The purpose of this paper is to demonstrate how to take full advantage of the advanced capabilities of the HP LaserJet in your production applications. I will accomplish this by demonstrating how my company has integrated the FANTASIA laser printing software into our internal sales and marketing system on the HP 3000.

PREMISE

The use of electronic forms only scratches the surface of how we can enhance our production reports with the LaserJet. Electronic forms are usually a static replacement for pre-printed forms. The use of these static electronic forms is a good first step. It can represent significant cost savings as well as added convenience with no changes to your source code.

The reference in the title of this paper to *dynamic* electronic forms means that you can incorporate typesetting technology into your production reports so that the forms you produce are dynamic – a form can adapt itself to the nature and amount of data that your application provides. Most importantly, this all takes place on the HP 3000 as part of your production runs. Typesetting technology allows us to go beyond electronic forms to produce much more effective laser output.

This paper focuses on the ability to *typeset* production data on the HP 3000 so that all of your reports have a professional, customized, typeset look. Although this technique involves changes to the source code of your application programs, the quality and effectiveness of the results easily justify the effort. In addition, I maintain that for new reports it is actually less work for programmer to take advantage of typesetting software than it is to produce a report in the more traditional manner.

BACKGROUND

Brant Technologies is a small company with a strong commitment to professionalism in all aspects of our business. In addition, we strive to be as highly automated and efficient as possible with minimized administrative overhead. The use of FANTASIA has helped us accomplish both goals.

The lead tracking and mail merge functions of our SPEEDWARE-based sales and marketing system had been evolving for several years as our needs evolved. The system incorporates an OMNIDEX interface to allow sales and marketing personnel to locate individuals or groups of people in our data base using many search criteria. Essentially the system contains contact information, prospect data for sales forecasting and notes on the history of client contact. We have since expanded this application into a sales system as well to generate invoices, commission statements, royalty reports, sales history and more. All reports are now produced with FANTASIA.

Giving a typeset, customized appearance to all of our printed output reflects well on the company. It makes us look professional while avoiding the costs associated with professionally typeset documents. But in addition, as soon as we enter an order to the system, we generate the form to order the product from our supplier, shipping documentation, an envelope (or label) and, most importantly, an invoice. There are no special forms to mount and specific output is directed to the appropriate LaserJet. All of this is driven by a salesperson entering a customer order. There is nothing to impede the process.

We will now examine the types of reports this system produces and what is involved in their production. I have also included some other examples I have developed to give the reader a more complete picture of the capabilities that are available.

REPORTS

Let's begin by taking a look at some of the output that our SPEEDWARE application produces.

Figure 1 is a typical invoice. This is the administration copy. You can see that we have had our logo scanned and we incorporate it into the invoice. In fact, the customer copy is printed on letterhead for the color and higher-quality paper. We print all subsequent copies with the scanned logo. Each copy has its destination identified at the bottom. The customer copy does not show the GL Code field so the Description field is wider.

Note that I am using proportional fonts throughout. The invoice has a much nicer look than the Courier or "typewriter" font of impact printers. In addition, the gap between the label and the data for Invoice Number, Purchase Order and Invoice Date is proportionately filled with dots for a customized look. Perhaps more importantly, the description field is typeset by FANTASIA and not by my application program. The program simply hands three 60-column fields of description data to FANTASIA and it is FANTASIA that decides how much will fit on a line and where to break the line. I no longer have to program that logic into my application. The same is true for Terms. The terms are "Net thirty days" and the Currency is "Canadian". My application gives FANTASIA four lines of data: the terms, the literal "in", the currency and the literal "funds.". FANTASIA puts these into a sentence and I am saved from having to format these in my application. So part of this form is static (ie. the grids and labels) and part is dynamic (ie. the data provided by the application).

If I had been using pre-printed forms, I could have replaced them very easily with "static" electronic forms, making no source changes to my application. Figure 2 shows what that would look like. Compare Figure 2 to Figure 1 and see which look you prefer. No matter which you choose, the use of electronic forms amounts to approximately 1/3 the cost of pre-printed forms, not including the extra costs associated with the initial design and setup of the pre-printed form, the waste, mounting, bursting, decollating, storing and bulk purchases of pre-printed forms.

There are also costs and time involved in form changes. Since implementing this system, we have had the GST (Goods and Services Tax) imposed in Canada and next year we will go through an area code change to our telephone number. Both of these changes have significant impact on forms. By using electronic forms I avoid the expense of ordering new forms and scrapping the old.

Figures 3 and 4 are examples of sales commission reports. The first, Figure 3, was one of the first forms I developed using FANTASIA. It looks very much like a replacement for a pre-printed form because the grids are static. Although it is a very attractive form, I have since changed the application to print Figure 4. This is truly a dynamic form where the data itself determines how large the grid area is. It easily spans multiple pages and the total box only appears at the end of the form. The most important data item on the page is the total commission paid and that is highlighted in a drop box with a larger font. We will take a look at the source code changes that I made to implement this report later in the paper.

Incidentally, this may not be a good example of a report which is cost justified to replace with enhanced laser technology. Typically, reports such as sales commission are internal and do not use pre-printed forms. But I maintain that all of your company's reports are important. If your report has a professional, customized look to it, it reflects well on the people who produced the report and it makes a difference to the recipient. Our sales people love their commission reports. It is important for them to feel that we appreciate their results and, quite frankly, this is a very easy report to produce.

Figure 5 is a fax message. I was unhappy with the quality of correspondence being faxed out of our office. We had a standard fax cover sheet to hand-write enclosure information, but these were usually illegible. Since most faxes are sent to people already in our data base (which has to be accessed anyway to check for the fax number) I added the capability to produce formatted faxes quite easily. Once the user selects the individual to whom he or she wants to address the fax, they select "FAX" from the report menu. This calls our favourite editor to allow the user to type the fax message. Once complete, the user simply exits from the editor and the system calls FANTASIA to process the FAX form and pull in their message text. It is simple, easy, fast and it gives a consistent, professional image to all material being faxed from our office.

Figure 6 is an example of archive output which I use solely to save paper. This shows a tape validate listing such as I keep for my backups. FANTASIA picks up all spool files named LIST (as well as SYSLIST, $STDLIST and others) and applies an "environment" file to print them four-up with boxes around each logical page. Note that I am only showing two pages up in portrait mode to fit in the proceedings. However the actual output appears in landscape mode with each of the boxes show taking up one quarter of the page. If I had a duplex printer, I could print on both sides of the paper which would give me eight logical pages on one physical piece of paper. I save paper, I save trees, and I take up less space in my binder of file listings. All of this adds up to saving money.

Figures 7 and 8 show before and after examples of a price list driven from a data base on the HP 3000. We do not publish price lists ourselves; this is an example produced for someone else. It is a good demonstration of how to incorporate typesetting into your production reports. Figure 7 shows the type of report that you can produce on the LaserJet using the default Courier font. Many companies produce catalogs and other reports that are sent out to be professionally typeset at considerable expense. Figure 8 demonstrates the enhanced output that could be created on the HP 3000 in a production environment. The nature of the data lends itself to being printed two-up in landscape mode. The header and footer data extend across the entire page. Perhaps more subtly, the footer area contains a number of terms, each of which is an entry in a TERMS data set. FANTASIA dynamically formats the footer area to leave enough room for all of the terms. So we get this professional look all driven from our production data and the formatter does much of the work for us.

My final example, Figure 9, shows a mail merge generating word processed-quality letters. These use "boiler plate" or standard text with variables such as name, address, salutation, etc. extracted from our corporate data bases. So the resulting letters all have the quality of a word processor but we are able to keep everything on the HP 3000 so that we can have large, production runs and treat them like any other report. Moreover, we can incorporate scanned signatures to save the time and expense of hand signing each letter.

This is not meant to be an exhaustive display of our enhanced reports; nor does it begin to cover all you can do with your LaserJet. However, it does represent a cross-section of the types of reports that we produce and some of the important output capabilities that many HP 3000 shops require.

INTEGRATION

Let's now take a look at how I have created two of these reports. Figure 10 shows a progression of my commission report from unformatted to simple formatting to the finished product. In order to accomplish this, I alter the source code to my application to embed FANTASIA formatting commands in with the data itself.

FANTASIA takes a post-processing approach. Your application creates output in the spooler or as flat files. FANTASIA then processes that output (calling "include" files, graphics, static forms, etc. as required) to create a spooler file of enhanced LaserJet output.

Figure 11 shows the file that produces the enhanced commission report in Figure 4. Note that it contains both data and FANTASIA commands – either a command line starting with a "\" in column 1 or delimited by "^" within data lines. The FANTASIA commands are created as literals in my application program. So the enhanced output can be created through Cobol, 4GLs such as SPEEDWARE and POWERHOUSE and even report writers which can include literals.

The body of the report is contained within the TABLE command. TABLE formats data into columns in a very flexible way. I have specified that the table contains 5 columns with column 2 a particular width. I specify the justification for the individual columns, shading if required and individual widths if I want to override the defaults. The actual data items can be separated by a specified delimiter, by at least two blanks, in specified column positions or one data item per line. The FANTASIA formatter will ensure that rows line up and it looks after breaking individual columns into multiple rows.

As a programmer, I have found that there are three important differences in creating output using FANTASIA over traditional methods (ignoring the quality and effectiveness of the output).

The first is that you have become married to your LaserJets. In most organizations that may not be an important point. But it is worth mentioning that if your LaserJet breaks, you can no longer redirect your FANTASIA output to your impact printers. It will not work.

On the plus side, the second difference is that the programmer now operates at a higher level. You no longer tell your application exactly how to print your data. You operate more as a typesetter and you instruct your application in how to process the data. If you want the data printed in a twelve point helvetica font, justified on both right and left margins and wrapped to fill fill all lines with 14 point line spacing in an area 3 inches wide measured 1 inch from the left margin you would issue the following commands:

```
\fontset h scale helv 12
\left 1;width 3;just both;font h;wrap;linespace 14
```

Programming is more productive because the FANTASIA formatter can do most of the work. FANTASIA looks after the logic involved in spacing and how much data can fit on each line, and so on. All we have to do is define the environment to FANTASIA and then give it the data.

My third point is that I now prototype my reports outside of my application development environment, be it SPEEDWARE, COBOL or whatever. I work with MPE files containing dummy data until I get the report to look exactly the way I want. Then I simply program my application to create output that looks like my prototype file. This is a real time saving because I do not have to go through the recompile, select, sort, etc. for each iteration of report refinement.

Figure 12 shows the MPE file containing the standard text and formatting directives to produce the mail merge letter in Figure 9. You can see how I have positioned and called the company logo at the end of the file. Although I have not included it here, I could have brought in a signature as well to produce ready-to-mail output.

This file contains formatting directives that define the fonts, margins, spacing, etc. In addition, it says that every time we see the macro "a" (as "^ma") in the letter, replace it with the next data item from the file "datafile" and repeat until we exhaust all the data in the file. This assumes that we have created an extraction file of of selected entries in the file "datafile". We could have done that with QUIZ, a report writer or any other means. We then run FANTASIA against this letter file to access that data and format the letter dynamically. This gives us a word processed look in our production environment on the HP 3000. And of course we can process one letter or thousands of letters once we have selected the appropriate data.

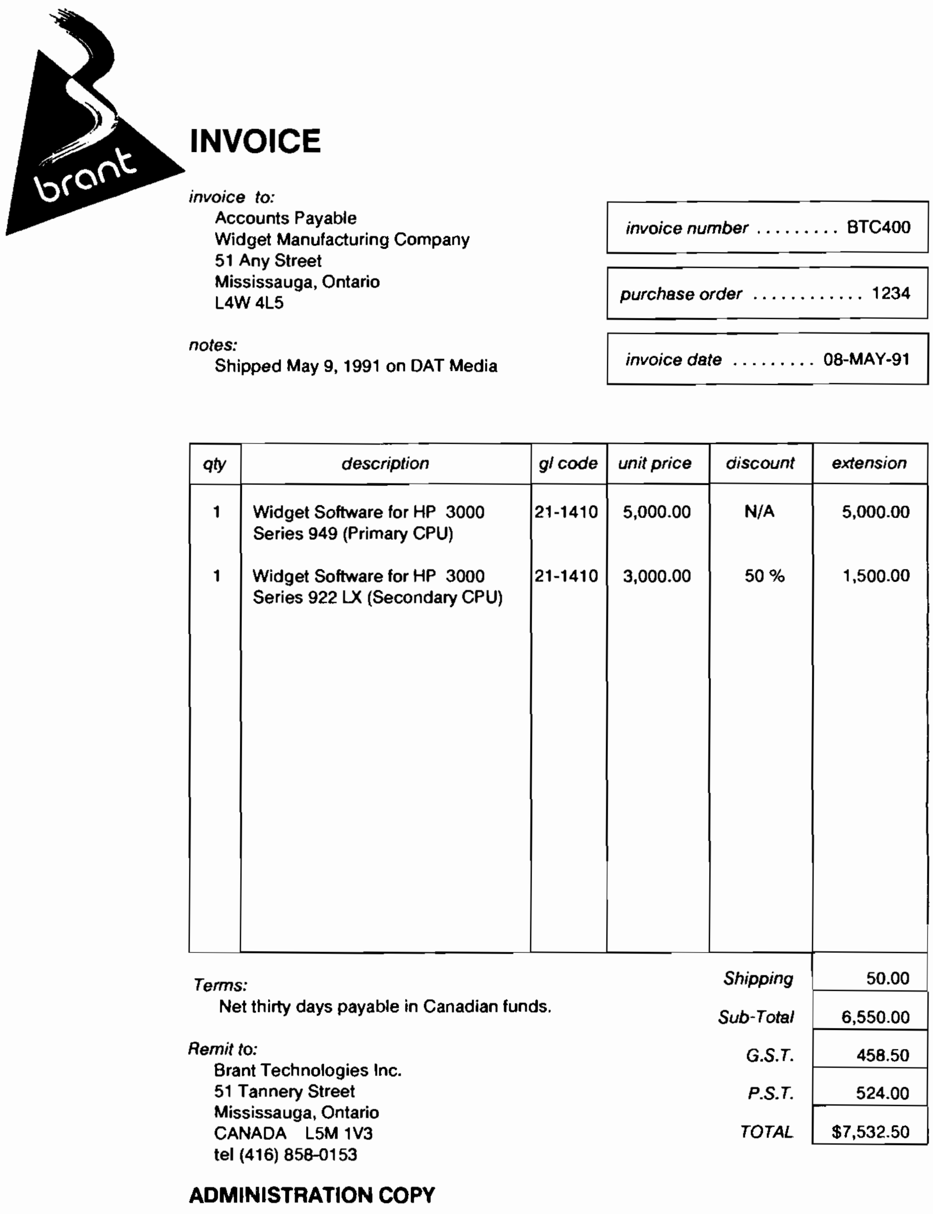
CONCLUSION

I have given you a taste of some of the enhanced LaserJet output we produce in our environment and have shown that electronic forms are much less expensive and much more convenient to use than pre-printed forms. Enhanced LaserJet reporting through the use of typesetting produces more effective reports that give our company a high-quality, professional image.

To make the best use of this technology, we made changes to the source code of our application programs. This is not necessary to simply replace pre-printed forms with static electronic forms. However, we made the source code changes to achieve a customized, typeset look in all of our producation reports. Although this represents an additional time investment, FANTASIA is not difficult to learn and we have been able to take advantage of the productivity aspects of the product.

The net result has been a dramatic improvement in the quality of our output. We produce better, more effective reports that reflect well on the whole organization.

Ross G. Hopmans
1992.06.09



INVOICE

invoice to:
Accounts Payable
Widget Manufacturing Company
51 Any Street
Mississauga, Ontario
L4W 4L5

invoice number	BTC400
purchase order	1234
invoice date	08-MAY-91

notes:
Shipped May 9, 1991 on DAT Media

qty	*description*	*gl code*	*unit price*	*discount*	*extension*
1	Widget Software for HP 3000 Series 949 (Primary CPU)	21-1410	5,000.00	N/A	5,000.00
1	Widget Software for HP 3000 Series 922 LX (Secondary CPU)	21-1410	3,000.00	50 %	1,500.00

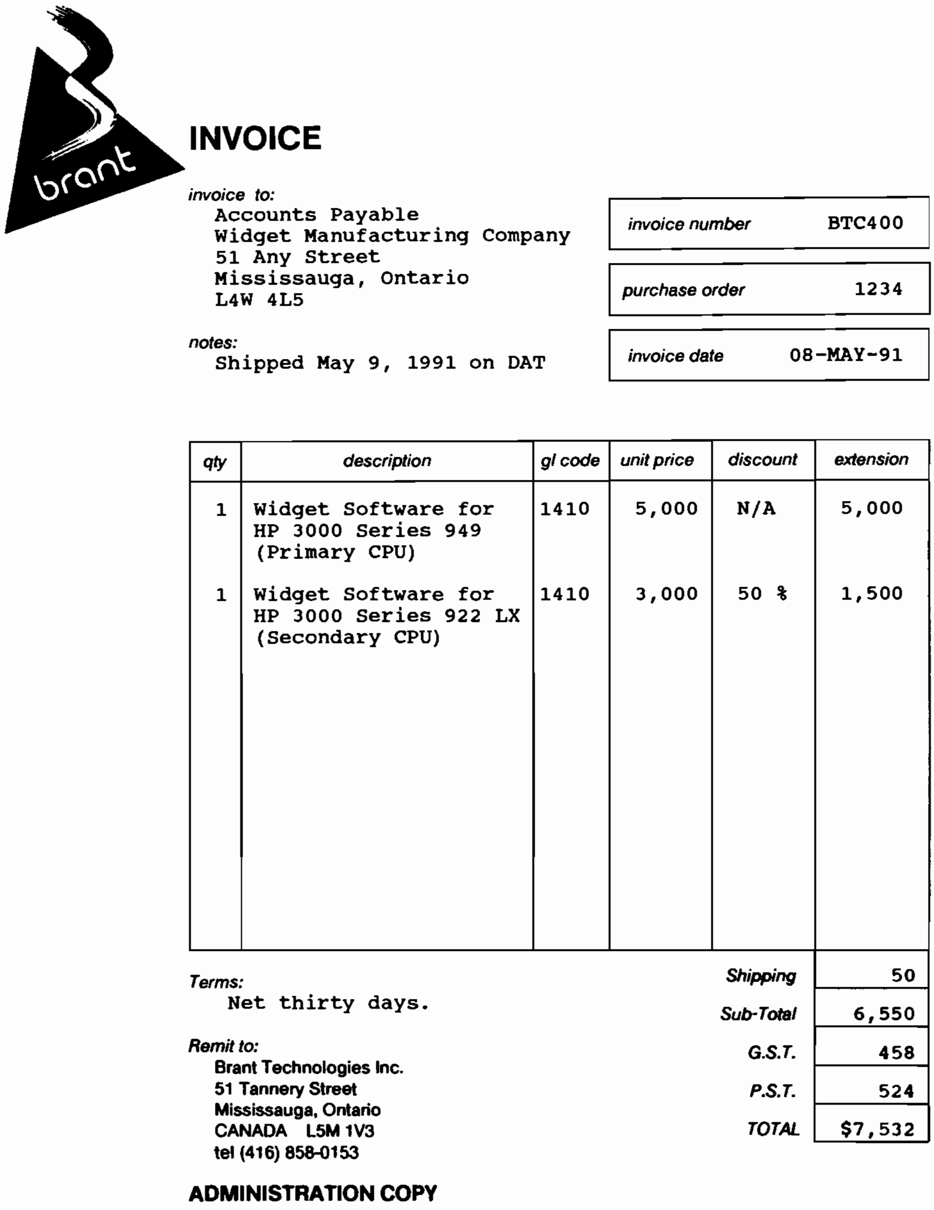
Shipping	50.00
Sub-Total	6,550.00
G.S.T.	458.50
P.S.T.	524.00
TOTAL	$7,532.50

Terms:
Net thirty days payable in Canadian funds.

Remit to:
Brant Technologies Inc.
51 Tannery Street
Mississauga, Ontario
CANADA L5M 1V3
tel (416) 858-0153

ADMINISTRATION COPY

Dynamic Electronic Forms – **FIGURE 1**



brant

INVOICE

invoice to:

Accounts Payable
Widget Manufacturing Company
51 Any Street
Mississauga, Ontario
L4W 4L5

notes:

Shipped May 9, 1991 on DAT

invoice number	BTC400
purchase order	1234
invoice date	08-MAY-91

qty	*description*	*gl code*	*unit price*	*discount*	*extension*
1	Widget Software for HP 3000 Series 949 (Primary CPU)	1410	5,000	N/A	5,000
1	Widget Software for HP 3000 Series 922 LX (Secondary CPU)	1410	3,000	50 %	1,500

Terms:

Net thirty days.

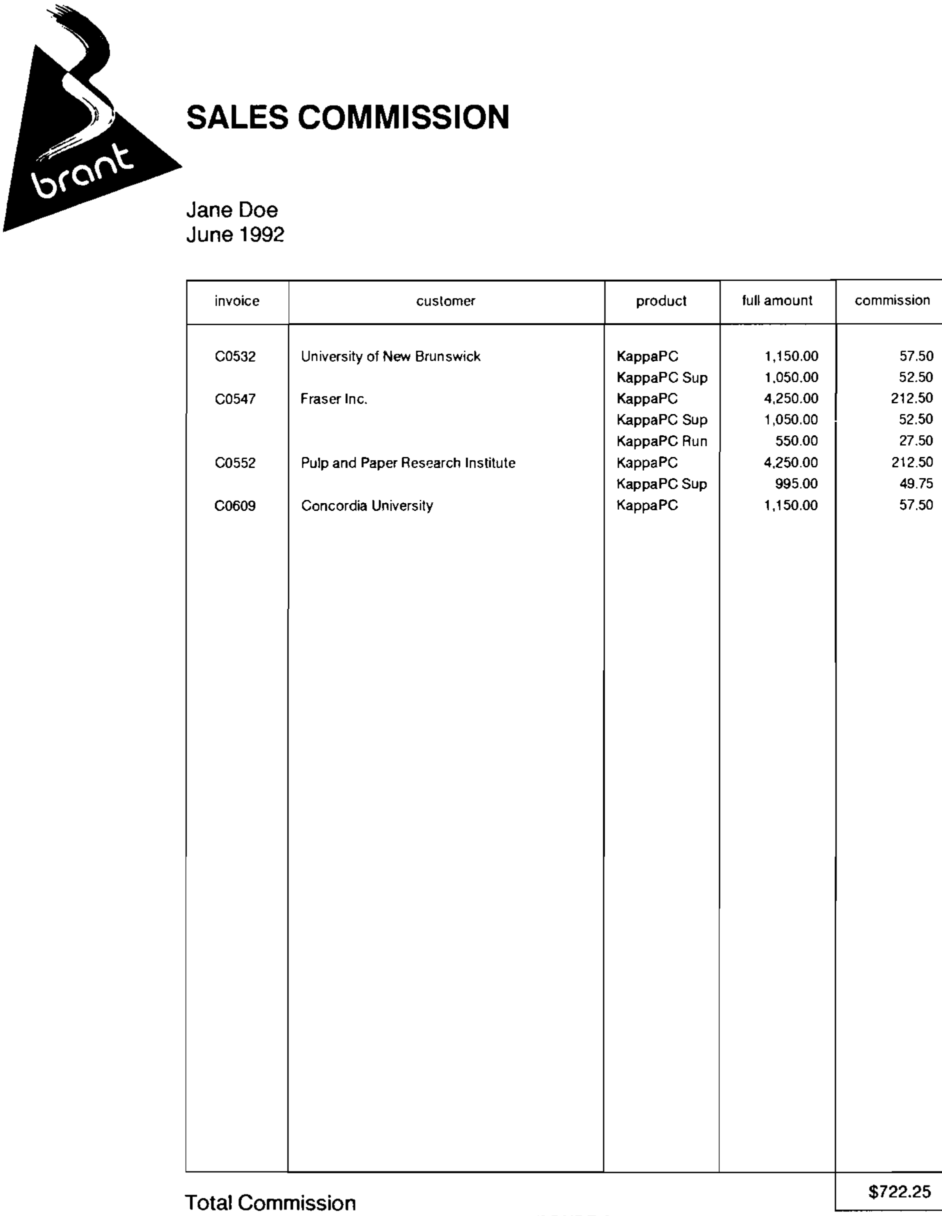
Shipping	50
Sub-Total	6,550
G.S.T.	458
P.S.T.	524
TOTAL	$7,532

Remit to:

Brant Technologies Inc.
51 Tannery Street
Mississauga, Ontario
CANADA L5M 1V3
tel (416) 858-0153

ADMINISTRATION COPY

Dynamic Electronic Forms – **FIGURE 2**



brant

SALES COMMISSION

Jane Doe
June 1992

invoice	customer	product	full amount	commission
C0532	University of New Brunswick	KappaPC	1,150.00	57.50
		KappaPC Sup	1,050.00	52.50
C0547	Fraser Inc.	KappaPC	4,250.00	212.50
		KappaPC Sup	1,050.00	52.50
		KappaPC Run	550.00	27.50
C0552	Pulp and Paper Research Institute	KappaPC	4,250.00	212.50
		KappaPC Sup	995.00	49.75
C0609	Concordia University	KappaPC	1,150.00	57.50

Total Commission $722.25

Dynamic Electronic Forms – **FIGURE 3**

BRANT TECHNOLOGIES INC.

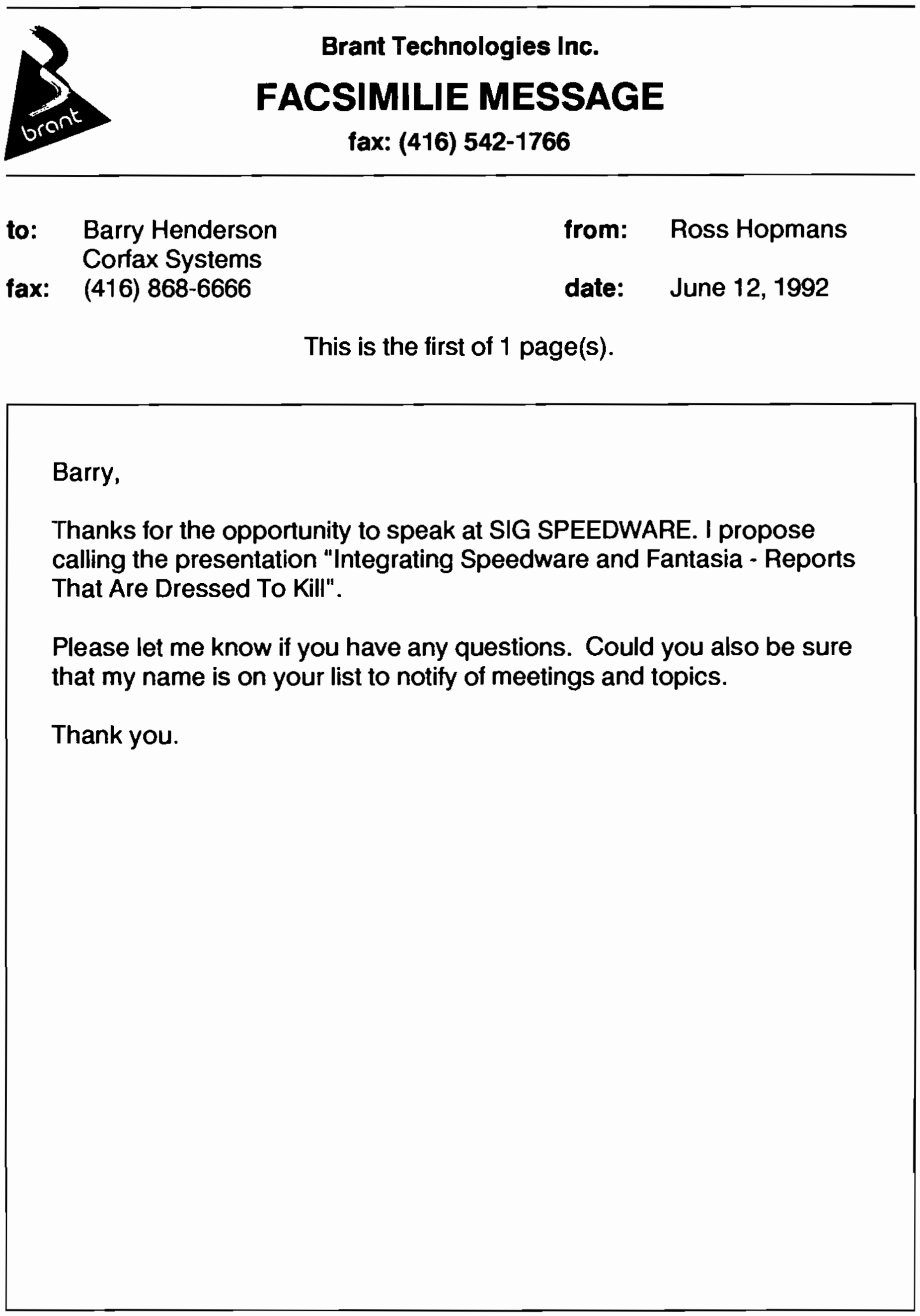
Commission Report - June 1992

Salesperson: **Jane Doe**

Invoice	Company	Product	Price	Comm
C0532	University of New Brunswick	KappaPC	1,150.00	57.50
		KappaPC Sup	1,050.00	52.50
C0547	Fraser Inc.	KappaPC	4,250.00	212.50
		KappaPC Sup	1,050.00	52.50
		KappaPC Run	550.00	27.50
C0552	Pulp and Paper Research Institute	KappaPC	4,250.00	212.50
		KappaPC Sup	995.00	49.75
C0609	Concordia University	KappaPC	1,150.00	57.50

$722.25

Dynamic Electronic Forms – **FIGURE 4**



Brant Technologies Inc.

FACSIMILIE MESSAGE

fax: (416) 542-1766

to:	Barry Henderson Corfax Systems	**from:**	Ross Hopmans
fax:	(416) 868-6666	**date:**	June 12, 1992

This is the first of 1 page(s).

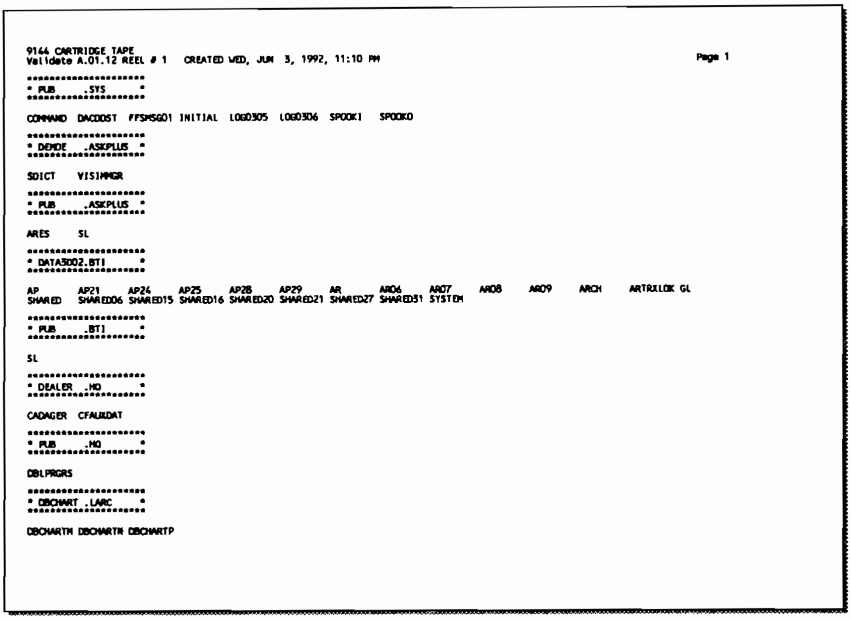
Barry,

Thanks for the opportunity to speak at SIG SPEEDWARE. I propose calling the presentation "Integrating Speedware and Fantasia - Reports That Are Dressed To Kill".

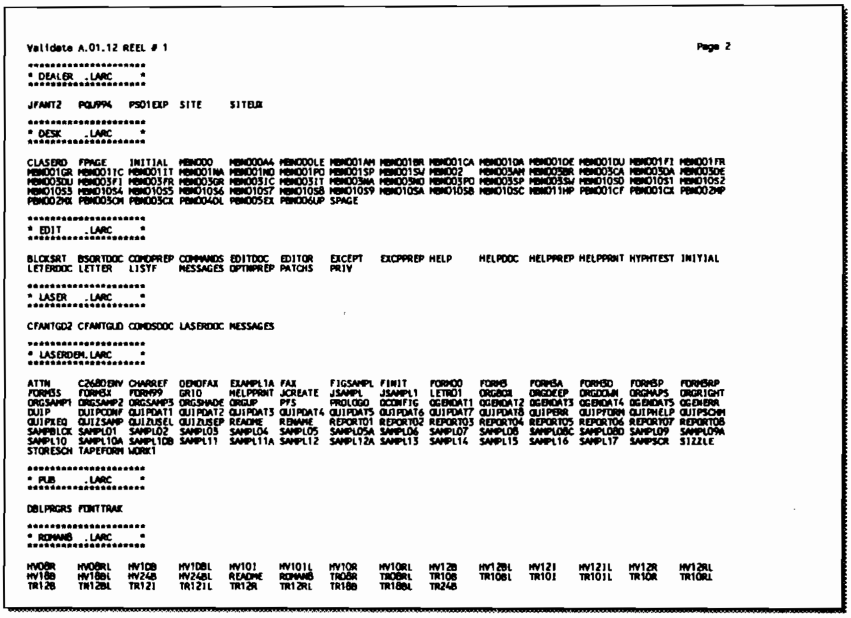
Please let me know if you have any questions. Could you also be sure that my name is on your list to notify of meetings and topics.

Thank you.

Dynamic Electronic Forms – **FIGURE 5**



```
9144 CARTRIDGE TAPE
Validate A.01.12 REEL # 1   CREATED WED, JUN  3, 1992, 11:10 PM                          Page 1

*********************
* PUB     .SYS      *
*********************

COMMAND  DACODST  FFSMSG01 INITIAL  LOG0305  LOG0306  SPOOKI   SPOOKO

*********************
* DEMOE   .ASKPLUS  *
*********************

SDICT    VISIMMGR

*********************
* PUB     .ASKPLUS  *
*********************

ARES     SL

*********************
* DATA3002.BTI      *
*********************

AP       AP21     AP24     AP25     AP28     AP29     AR       AR06     AR07     AR08     AR09     ARCH     ARTRXLOK GL
SHARED   SHARED06 SHARED15 SHARED16 SHARED20 SHARED21 SHARED27 SHARED31 SYSTEM

*********************
* PUB     .BTI      *
*********************

SL

*********************
* DEALER  .HO       *
*********************

CADAGER  CFAUXDAT

*********************
* PUB     .HO       *
*********************

DBLPRGRS

*********************
* DBCHART .LARC     *
*********************

DBCHARTM DBCHARTN DBCHARTP
```



```
Validate A.01.12 REEL # 1                                                                 Page 2

*********************
* DEALER  .LARC     *
*********************

JFANT2   PQUP94   PS01EXP  SITE     SITEUX

*********************
* DESK    .LARC     *
*********************

CLASERD  FPAGE    INITIAL  MEN000   MEN000A4 MEN000LE MEN001AM MEN001BR MEN001CA MEN001DA MEN001DE MEN001DU MEN001FI MEN001FR
MEN001GR MEN001IC MEN001IT MEN001NA MEN001NO MEN001PO MEN001SP MEN001SW MEN002   MEN003AM MEN003BR MEN003CA MEN003DA MEN003DE
MEN003DU MEN003FI MEN003FR MEN003GR MEN003IC MEN003IT MEN003NA MEN003NO MEN003PO MEN003SP MEN003SW MEN010S0 MEN010S1 MEN010S2
MEN010S3 MEN010S4 MEN010S5 MEN010S6 MEN010S7 MEN010S8 MEN010S9 MEN010SA MEN010SB MEN010SC MEN011HP PBN001CF PBN001CX PBN002MP
PBN002MX PBN003CM PBN003CX PBN004OL PBN005EX PBN006UP SPAGE

*********************
* EDIT    .LARC     *
*********************

BLCKSRT  BSORTDOC CONDPREP COMMANDS EDITDOC  EDITOR   EXCEPT   EXCPPREP HELP     HELPDOC  HELPPREP HELPPRNT HYPHTEST INITIAL
LETERDOC LETTER   LISYF    MESSAGES OPTNPREP PATCHS   PRIV

*********************
* LASER   .LARC     *
*********************

CFANTGD2 CFANTGUD COMDSDOC LASERDOC MESSAGES

*********************
* LASERDEM.LARC     *
*********************

ATTN     C2680ENV CHARREF  DEMOFAX  EXAMPL1A FAX      FIGSAMPL FINIT    FORM00   FORM5    FORM5A   FORM5D   FORM5P   FORM5RP
FORM5S   FORM5X   FORM99   GR10     HELPPRNT JCREATE  JSAMPL   JSAMPL1  LETR01   ORGBOX   ORGDEEP  ORGDOWN  ORGMAPS  ORGRIGHT
ORGSAMP1 ORGSAMP2 ORGSAMP3 ORGSHADE ORGUP    PFS      PROLOGO  QCONFIG  QGENDAT1 QGENDAT2 QGENDAT3 QGENDAT4 QGENDAT5 QGENERR
QUIP     QUIPCONF QUIPDAT1 QUIPDAT2 QUIPDAT3 QUIPDAT4 QUIPDAT5 QUIPDAT6 QUIPDAT7 QUIPDAT8 QUIPERR  QUIPFORM QUIPHELP QUIPSCHM
QUIPXEQ  QUIZSAMP QUIZUSEL QUIZUSEP README   RENAME   REPORT01 REPORT02 REPORT03 REPORT04 REPORT05 REPORT06 REPORT07 REPORT08
SAMPBLCK SAMPL01  SAMPL02  SAMPL03  SAMPL04  SAMPL05  SAMPL05A SAMPL06  SAMPL07  SAMPL08  SAMPL08C SAMPL08D SAMPL09  SAMPL09A
SAMPL10  SAMPL10A SAMPL10B SAMPL11  SAMPL11A SAMPL12  SAMPL12A SAMPL13  SAMPL14  SAMPL15  SAMPL16  SAMPL17  SAMPSCR  SIZZLE
STORESCH TAPEFORM WORK1

*********************
* PUB     .LARC     *
*********************

DBLPRGRS FONTTRAK

*********************
* ROMAN8  .LARC     *
*********************

HV08R    HV08RL   HV10B    HV10BL   HV10I    HV10IL   HV10R    HV10RL   HV12B    HV12BL   HV12I    HV12IL   HV12R    HV12RL
HV18B    HV18BL   HV24B    HV24BL   README   ROMAN8   TR08R    TR08RL   TR10B    TR10BL   TR10I    TR10IL   TR10R    TR10RL
TR12B    TR12BL   TR12I    TR12IL   TR12R    TR12RL   TR18B    TR18BL   TR24B
```

Dynamic Electronic Forms – **FIGURE 6**

```
                  Widget Software Products Inc.

Effective April 9, 1991                                    Page 1
-----------------------------------------------------------------------
Product Description                                     Price ($)

Language Compilers

100348   Pascal Compiler for IBM PC Compatibles            340.00
100352   Pascal Forms Management Utility                   125.00
101008   Cobol Compiler                                    555.00
101009   Cobol Forms Management Utility                     75.50
102044   Basic Interpreter                                 390.00
102046   Basic Compiler                                    450.00
102047   Basic Compiler Run Time System                    199.00
102048   Basic Compiler Forms Management                   240.00
103031   Fortran Compiler                                  350.00
103032   Fortran Graphic Extensions                        199.00
103033   Fortran Language Extensions                        75.00

Data Base Management Systems

112345   Superbase Relational DBMS                         670.00
112348   Superbase Run Time System                         230.00
112349   Superbase Management Utility                      145.50
112350   Superbase Forms Interface Compiler                340.00
112351   Superbase Report Writer                           265.50
112352   Superbase SQL Language Module                     288.00
112353   Superbase Run Time Optimiser                      350.00
112354   Superbase Extended System Option 1                199.00
112355   Superbase Extended System Option 2                199.00
112356   Superbase Extended System Option 3                199.00
112357   Superbase Reconfiguration System                   75.60
113011   Fastbase Network DBMS                             475.00
113012   Fastbase Distributed System Option 1              240.00
112348   Superbase Run Time System                         230.00
112349   Superbase Management Utility                      145.50
112350   Superbase Forms Interface Compiler                340.00
112351   Superbase Report Writer                           265.50
112352   Superbase SQL Language Module                     288.00
112353   Superbase Run Time Optimiser                      350.00
112354   Superbase Extended System Option 1                199.00
112355   Superbase Extended System Option 2                199.00
112356   Superbase Extended System Option 3                199.00
```

Dynamic Electronic Forms – **FIGURE 7**

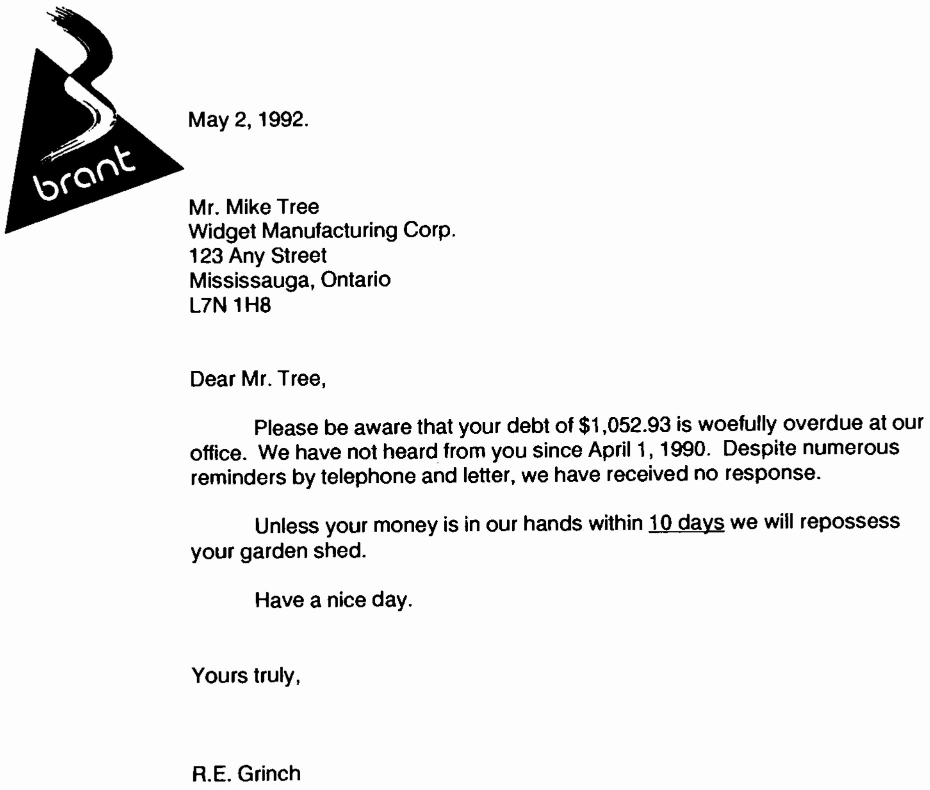
Widget Software Products Inc.

Effective April 9, 1991 *Page 1*

Product No.	Description	Price ($)
Language Compilers		
100348	Pascal Compiler for IBM Compatibles	340.00
100352	Pascal Forms Management Utility	125.00
101008	Cobol Compiler	555.00
101009	Cobol Forms Management Utility	75.50
102044	Basic Interpreter	390.00
102046	Basic Compiler	450.00
102047	Basic Compiler Run Time System	99.00
102048	Basic Compiler Forms Management	240.00
103031	Fortran Compiler	350.00
103032	Fortran Graphic Extensions	99.00
103033	Fortran Language Extensions	75.00
Data Base Management Systems		
112345	Superbase Relational DBMS	670.00
112348	Superbase Run Time System	230.00
112349	Superbase Management Utility	145.50
112350	Superbase Forms Interface Compiler	340.00
112351	Superbase Report Writer	265.50
112352	Superbase SQL Language Module	288.00
112353	Superbase Run Time Optimiser	350.00
112354	Superbase Extended System Option 1	199.00
112355	Superbase Extended System Option 2	199.00
112356	Superbase Extended System Option 3	199.00
112357	Superbase Reconfiguration System	75.60
113011	Fastbase Network DBMS	475.00
113012	Fastbase Distributed System Option 1	240.00
113013	Fastbase Distributed System Option 2	240.00
114012	Superbase/2 Relational DBMS	670.00
114013	Superbase/2 Run Time System	230.00
114014	Superbase/2 Management Utility	145.50
114350	Superbase/2 Forms Interface Compiler	340.00
114351	Superbase/2 Report Writer	265.50
114352	Superbase/2 SQL Language Module	288.00
114353	Superbase/2 Run Time Optimiser	350.00
114354	Superbase/2 Extended System Option 1	199.00
114355	Superbase/2 Extended System Option 2	199.00
114356	Superbase/2 Extended System Option 3	199.00
114357	Superbase/2 Reconfiguration System	75.60
115011	Fastbase/2 Network DBMS	475.00
115012	Fastbase/2 Distributed System Option 1	240.00
115013	Fastbase/2 Distributed System Option 2	240.00

Prices are subject to change without notice. Prices include delivery. Contact your sales representative for a specific quotation.

Dynamic Electronic Forms – **FIGURE 8**



brant

May 2, 1992.

Mr. Mike Tree
Widget Manufacturing Corp.
123 Any Street
Mississauga, Ontario
L7N 1H8

Dear Mr. Tree,

Please be aware that your debt of $1,052.93 is woefully overdue at our office. We have not heard from you since April 1, 1990. Despite numerous reminders by telephone and letter, we have received no response.

Unless your money is in our hands within 10 days we will repossess your garden shed.

Have a nice day.

Yours truly,

R.E. Grinch

Dynamic Electronic Forms – **FIGURE 9**

```
                 WIDGET SOFTWARE PRODUCTS
              Commission Report - April 1991

Salesperson:  Jane Doe

Invoice Company              Product      Price      Comm
------- -------------------- ---------- -------- ---------
ABC123  First National Bank  Widgets     15,000     750.00
ABC124  Jim's Garage         Widgets     23,000   1,150.00
ABC124  Jim's Garage         Tie Rods     7,500     375.00
ABC125  Orange County        I Beams      1,200      60.00
ABC125  Orange County        Widgets     14,500     725.00

        TOTAL                                    $3,060.00
                                                 =========
```

WIDGET SOFTWARE PRODUCTS

Commission Report - April 1991

Salesperson: Jane Doe

Invoice	Company	Product	Price	Comm
ABC123	First National Bank	Widgets	15,000	750.00
ABC124	Jim's Garage	Widgets	23,000	1,150.00
ABC124	Jim's Garage	Tie Rods	7,500	375.00
ABC125	Orange County	I Beams	1,200	60.00
ABC125	Orange County	Widgets	14,500	725.00

WIDGET SOFTWARE PRODUCTS

Commission Report - April 1991

Salesperson: **Jane Doe**

Invoice	**Company**	**Product**	**Price**	**Comm**
ABC123	First National Bank	Widgets	15,000	750.00
ABC124	Jim's Garage	Widgets Tie Rods	23,000 7,500	1,150.00 375.00
ABC125	Orange County	I Beams Widgets	1,200 14,500	60.00 725.00

$3,060.00

Dynamic Electronic Forms – **FIGURE 10**

```
\port; left 1; width 6; nowrap; just left; top 1.55
\enspace |; special #; expand @; thick 3; shade 20
\linespace 9; just center
\fontset t 19, r 5, i 15, b 16
\* fontset t scale helv 18 bold
\* fontset r scale times 10
\* fontset i scale helv 10 ital
\* fontset b scale helv 10 bold
^ft^BRANT TECHNOLOGIES INC.

^fb^Commission Report - June 1992

\just left
^fi^Salesperson:  ^fb^Jane Doe
\table 5 just left col 1 just center col 2 width 2.2 horiz 0 &
\table col 4 just right col 5 just right col 5 shade 2 &
\table col 1 width 0.6 col 3 width 1.0
\fillbox 0.25; font b
^nc^Invoice  ^nc^Company  ^nc^Product  ^nc^Price  ^nc^Comm
\line 8; font 14
C0532  University of New Brunswick        KappaPC      1,150.00   57.50
|      |                                  KappaPC Sup  1,050.00   52.50
\line 8
C0547  Fraser Inc.                        KappaPC      4,250.00  212.50
|      |                                  KappaPC Sup  1,050.00   52.50
|      |                                  KappaPC Run    550.00   27.50
\line 8
C0552  Pulp and Paper Research Institute  KappaPC      4,250.00  212.50
|      |                                  KappaPC Sup    995.00   49.75
\line 8
C0609  Concordia University               KappaPC      1,150.00   57.50
\tableend
\inleft 5.05; shade 50
\dropbox 0.4 0.1

\space inch 0.1; just center; font b
$722.25
```

```
\left 1.5; width 5; top 2; just left; nowrap; font 8
\fillin macro=a, file=datafile, repeat
^d1^.

^ma
^ma
^ma
^ma
^ma
\space inch 0.25
Dear ^ma,

\indent 0.5; wrap
Please be aware that your debt of ^ma is woefully
overdue at our office.  We have not heard from you
since ^ma.  Despite numerous reminders by
telephone and letter, we have received no
response.

Unless your money is in our hands within ^u10 days^s
we will repossess your ^ma.

Have a nice day.

\indent off; nowrap
Yours truly,



R.E. Grinch

\goto 0.16 0.1
\graph brant1p.pub
```

Dynamic Electronic Forms – **FIGURE 12**

APPLICATION INSTALLATION

FRANNIE CASELLA
NORTHERN CALIFORNIA CANCER CENTER
32960 ALVARADO-NILES ROAD, SUITE 600
UNION CITY, CA 94587
(510) 429-2531

OVERVIEW

This paper will address the issues and constraints in installing third party maintained software on an HP3000. The software mentioned and the terminology used are specific to the HP environment, yet the concepts can easily be applied to any other system. Just for reference, we are running an HP925 with version 3.0 of MPE/XL.

Our organization has the source code to our major application but the bulk of the maintenance of this application is carried out by another organization which owns the source code. We also install our own modifications to this source code due to our own special needs. Thus the nightmare begins.

INTRODUCTION

At one time or another we have all been a slave to two masters. If you are lucky, the two want very similar things from you. If you are not so lucky, you end up writing a paper like this.

First a brief background on my two masters. The first and primary master is the Federal Government; specifically the National Cancer Institute (NCI). My second one is the State of California; specifically the Department of Health Services. We hold contracts from each to periodically provide them with specific data from our Cancer Tumor Database. Our application software

CANDIS (CANcer Data Information System), which is used to maintain the data, is owned by the State and the bulk of the maintenance is carried out by them.

We collect cancer data from all hospitals in the nine San Francisco Bay Area Counties using a PC software product named C/NET[2]. This software is also maintained by the State but is totally independent of the CANDIS software. The data is uploaded, edited and reported on bi-monthly by the CANDIS software through a series of COBOL programs which run in batch mode. The data is checked for accuracy and completeness and then modified through a series of VPLUS screens and three COBOL programs which run on-line.

The cancer data is then regularly sent to both the State and the Feds, but each requires the data in a different format. For this reason, we enhance the State's software to comply with the standards of the Federal Government to handle those situations when the requirements for the Feds can be easily incorporated into the State's software. Confused? Join the club!!

The standard we follow is that the database holds the data according to the State's specifications. Any modification which is necessary for the Feds but not the State is then handled in a separate program. This program runs on the data after it has been extracted from the database. Any recodes to data or any manipulation which needs to be made to conform to the Federal regulations is done to the extract file only. This way the database stays in sync with the State and the Feds receive their data in the format they stipulate. This keeps both masters happy.

BACKGROUND

After the State installed the first version of CANDIS, we made the modifications necessary to meet the Federal requirements that wouldn't interfere with the State's specifications. The big fiasco didn't happen until the second version release came from the State. After the installation we found that several of the modifications we had made for the Fed's were gone. After making all of these changes AGAIN, we realized that it was time to sit down and plan our strategy before the next release. Neither the programming staff nor the users wanted to ever go through another few weeks like we had after that last installation.

The following is a list of issues which we put together as a checklist of sorts to make sure that the next installation went quickly, smoothly and, most important, that nothing was forgotten.

- receiving the software
- the accounting structure
- comparing the new release with the old
- modifications
- keeping track
- testing
- installing

Receiving the Software

This seems like a somewhat trivial point to bring up but what it includes is the fact that you should be prepared for the software prior to physically receiving the release.

Try to get an exact date of the release if this is at all possible. Many times it is not. Try to pin down the date to something a little more precise than "spring". At least go for a specific month and once you have that you can usually determine if it will be early in the month or more towards the end. This is important for your initial planning to determine when the software will be available to the end users once the release is in-house.

One must remember that the end users will be impacted by this version release also. If there are major changes perhaps training will have to be arranged. In our shop, we are on a fixed production schedule. We upload approximately 2500 to 3000 new records every two weeks.

It is imperative that we keep on this schedule to insure that we stay current and don't let the cases get back-logged. What this means is that we must coordinate the installation with the production calendar. We try to minimize the user down time which usually means working on the weekends.

If you take some time prior to receiving the release to prepare for its installation you can save yourself time and energy. I know this is a new concept for some of you, it was for us. We call it Planning Ahead!!

If you have a large programming staff take the time early to pick the lucky soul(s) who will be doing the install. You might even want to let them know ahead of time so that they can be mentally prepared for it. If your staff is small and the person knows who he or she is, you might want to just sneak up on them, smile politely, whisper the install date and then run. This gives them time to finish current projects and adjust their workload accordingly.

An important and somewhat time-consuming task is creating an "install" environment. This should be created as similar to the production environment as possible. If room permits on the system, one can just store the production account and restore it into a test account. If you are like most of us, space on your system is a bit tight. What we did was to create two test databases. The first is very small, approximately 500 records. It is used for all the initial testing. It is by no means a perfect test database. It does allow the programmers to quickly test out their programs to see if they run to completion. It also lets them work out any bugs in their JCL, e.g. missing file statements, incorrect file names, etc. It is strictly used as a primary check before any thorough testing begins.

We then created a scaled-down version of the database as close to production as possible. A product such as SUPRTOOL by Robelle is great for this. It allows you to select every "nth" record from a dataset into a flat file and then will even load these records into a test database with very little effort. SUPRTOOL's table function will even hold the keys to these selected records so that all associated records from other datasets can be extracted, thus creating a complete test database.

This can all be done ahead of time so that when the software does arrive it can be immediately put onto the system and nobody has to wonder "Now where did I put that tape?".

Another task which can be completed prior to the arrival of the tape is a checklist to help keep track of each program, module, copylib, JCL stream, etc. Just print each of these down the left side of a sheet of paper. Print them in some order that makes it easiest for you. Either alphabetically, by system and then subsystem, or even in the order in which they are run. Whatever makes the most sense to you. Next, draw vertical lines down the page giving you many columns. As we go along we will fill in the columns. The column headings I use are:

- compared/changes?
- possible problem
- moved into SOURCE
- modifications made
- clean compile and prep (or link)
- modular test
- unit test
- system test
- moved into production
- moved into library
- bugs
  - found
  - reported to State
  - resolved

Use as many pages as you need, but make sure they are numbered 1 of n, 2 of n, etc. to be sure that no pages get misplaced and that everything is accounted for. Leave empty space at the end to add the names of new programs which might come along with the new release. You may even want to tape two pieces of paper together in order to give yourself wider columns with more room to mark the columns and to jot down notes.

THE ACCOUNTING STRUCTURE

As mentioned in the previous section, it is very important to keep your test environment as close to your production environment as possible. Of course, we all know this already. What is also important though is keeping the current software totally separate from the new version release. This can get quite confusing if you aren't careful. Don't mix any new and old software until the actual install. The way we handled this was to set up three groups in the test account. We call them SRCOLD, SRCNEW and SOURCE.

SRCOLD holds all of the current source code (this can be restored from your Library account or wherever your current software is kept); SRCNEW is where the new release is put; and the SOURCE group is where the newly modified version will be assembled.

The same holds true for the JCL. When receiving updates for running programs in batch mode, we also set up groups to hold these files. We use JOBOLD, JOBNEW and JOBS. The same logic applies as above for the source code. Once the accounting structure is set up and the new release downloaded onto the HP3000, you are ready to start looking at just what it is that you have been sent.

An often overlooked detail is making sure that the install environment has the identical ACCESS and CAPABILITIES as the production environment. You want to ensure that when the application is moved into production that it runs without a hitch.

COMPARING THE NEW RELEASE WITH THE OLD

I don't know about most of you out there but when I install a new version of software I like to know exactly what changes have been made to it before the users come running down the hall screaming "Did you know that this program now does this?". This way I can always say "Yes, of course I do -- I installed it". If you have thoroughly reviewed the code there should never be any surprises.

More importantly, by looking at the modifications that have been added to the code <u>before</u> you install them for the users, you have a chance to discuss them with the users and determine whether you, in fact, should install the change. It really does pay in the long run to take the time and discuss modifications with the people who have the authority to decide whether the application would be better off with or without the new change. This not only makes it easier on the programmer who then only has to install it once, but it also makes the users feel that they have some input into the application they are using.

Hopefully, your release comes with some kind of documentation as to which programs have been modified and which have not. This was not always the case for us. Eventually we did start getting some documentation but we had already come up with a way of determining the changes ourselves. This is where our checklist comes in. The first column is titled "compared/any changes?". Each new program, module, copylib, or JCL etc., is compared against the current version. As we review each piece, it is checked off as having been looked at. A notation of "Y" or "N" is made in the column as to whether or not any code has been modified.

A great product for comparing different versions of source code is SCOMPARE by the ALDON Computer Group. What this software does is to compare two versions of the same program and detail every line of code which has been modified. The output is very easy to comprehend. You even have the option of creating an editfile. This file is created at the time of the compare and holds all the line numbers of which should be added and which should be deleted to make the old version match up with the new version.

If you decide that the new version is not quite what you want to install, SCOMPARE allows you to pick and choose which new lines of code you do want and easily discard the lines you don't, via the editfile. You then "use" the file in conjunction with the EDITOR and it will insert and delete the corresponding line numbers for you. There is no typing on your part, thus removing the possibility of mistakes.

I don't know of any other software on the HP3000 that does anything similar to this, but I can say that if you need to compare modules or entire programs SCOMPARE can easily save you hours of work. SCOMPARE will even do all the members in your copylibs as well as the comparison between old and new JCL streams.

One standard we follow is that the changes we receive but do not wish to install are left in the program and just commented out. This way the code stays in the program as self-documentation and is available if the users change their minds in the future.

At this point we are not only looking at the code that was added but looking for code that may not have been added. What I mean is that if the documentation states that code has been added to perform a specific function, I look to see that somewhere there is code to do that. This also goes for bug fixes. Has the code been modified that is going to take care of all the bugs listed as "fixed" in the documentation? Has any code been removed which shouldn't have been? It is just as important to find the code which has been removed as it is to check the code which has been added.

I check off all these types of things on my checklist under the heading "possible problem" by marking a "Y" or "N". When all of the pieces have been looked at, your checklist can be reviewed to help in determining the amount of time it will take to do the install. This checklist then becomes part of the documentation of the installation and is kept for future reference. I also attach notes that I have made regarding the reason modifications were made or not and who made these decisions.

MODIFICATIONS

Once the modifications have been identified and everyone has agreed on which ones to install, the process is very simple. Move a copy of the new source code from the SRCNEW group into SOURCE. If no modifications are necessary, move the next program in, etc. If modifications are needed, do them in the SOURCE group leaving the SRCNEW group as it originally came in.

This separation is important in case you need to start over again with modifications to a particular program or if you need to go back and look at the original code for any reason. This SOURCE group will then eventually hold all the programs for your new release and can be easily moved into either your production or library account.

Be sure to put documentation into the programs which you modify stating your name, date and the modifications made. This makes it much simpler for anyone coming in later to make modifications. They will know right away by looking at the program that it has had in-house modifications made to it. This may or may not affect the revisions they need to make but it will alert them to that possibility.

The idea here is to keep the three versions all together in one account but in three separate groups. This may seem like a lot of unnecessary work but in the long run makes things much simpler. When the time comes to move the entire release into production, the library and off to tape for safe keeping, it is much easier to deal with an entire group rather than moving some from one group and a few from another.

Once the changes have been verified, I do my compile and prep (or link). Again, here I keep track by marking a few more columns: "moved into Source", "modifications made", and "clean compile and prep". If all goes well, I check it off and move onto the next one until they're all done.

KEEPING TRACK

Keeping track of which programs have been revised can be as simple as the checklist I mentioned previously. In a small shop like ours I don't have the luxury of closing my door while working on a project like this. I still have to be there to answer questions for users and to do ad hoc troubleshooting at a moments notice. If you work in an environment like this where you are constantly being interrupted, this simple checklist can save your sanity. To tell you the truth, even without interruptions from users I sometimes find it hard to remember what I did yesterday, so you can see why this is a must for me!

TESTING

I am not going to go into the fine art of testing here but I do want to stress the importance of testing each of the programs which have been modified. Even the smallest change to a piece of code can make HUGE errors, as I'm sure you are all aware. We once had a programmer working for us whose answer to the question "did you test it?" was "well, just look at the code - it's only one line". Not a good answer!!! Even the one-line changes need to be fully tested.

As I mentioned earlier in this paper and I'm sure all you programmers out there already know, it is imperative that your testing environment be as close to the real thing as possible. Many bugs that slip through are because the testing environment does not accurately represent what is out there on the live database.

Also remember that some things that need to be tested are codes not found in the live database but which are tested for in the programs. You need records that are old as well as more recent. You need some with multiple entries (if this is possible for you) and those that are single. You need a wide variety of codes in different fields so that all possibilities are checked for each new piece of code.

If the record does not exist on the test database which will thoroughly test a piece of code, I use QUERY or DBEDIT from Robelle to create the record or to modify one that will force it to fit my specifications. Just remember that taking the time to test thoroughly up front prevents a lot of headaches later on down the road.

After each of the modified pieces of code has been tested, I then test it again within the context of the entire program. I select certain records on the test database which I follow throughout the program. I then check the end results to be sure they match my expected results. When this has been done for all programs, I then run a complete application test to verify that all the programs work together the way they were intended. Here again, a few more columns to mark off on the checklist, "modular test", "unit test" and "system test".

Testing is very time consuming. I don't let this portion be hurried. I make sure that when we are targeting an installation date that I have given myself plenty of time for testing. If things don't go well when testing, we delay the install date rather than put untested code into production.

INSTALLING

The actual installation of the new software should now be a piece of cake. All of the new code is together in one group, it has all been tested and now it needs to be moved into the production environment. This can be as simple as storing your executable code from the test account to tape and restoring it into the production account. The same can be done for the source code (if it is to be kept in the production account).

If you have the MPEX software by VESOFT, copying the files from account to account is even easier. Providing you have the correct capabilities, one simple command will allow you to copy or move an entire group from your test group into the production account.

We have a "library" account which holds all the new source code as well as the previous production version as a backup. This guarantees that if revisions need to be made at a later date, the programmer knows where the current version resides. There should never be any doubt as to which program is the version to be modified!!! We also keep a copy of the original source code, as it came from the State, stored off to tape.

After the source code is in the "library" account (or where ever it is kept on your system) it is a very good idea to back it all up onto tape. Then all of the source should be removed from the test environment. This removes the possibility of someone assuming that the code in "test" is the current version and using it when making changes in the future. That is a very easy way to get your software out of sync. Use your checklist to document that the code has been either moved to production or to the library account under the headings "moved into production" and "moved into library".

For those programs which will be running in batch mode, be careful that the job card has been modified to point to the production account. Another gotcha here is to be sure that the output files for production are built to the proper size. It's very easy to forget to increase their capacities after they have been built for testing in the test account. This is more of an issue for those of you on a classic machine than for those on an XL but considering that when I originally wrote this paper I was on a classic, I thought I'd mention it.

Again, be sure that the users are aware when the installation is to occur. If it is to happen over the weekend make certain that people who occasionally or regularly work on the weekend know that the install is taking place and that the production account will be inaccessible. We have made it a policy that when we are installing new software in the production account that the entire system is off limits to everyone. This saves time in trying to figure out who needs to know and who doesn't. <u>Everyone</u> needs to know and <u>everyone</u> needs to stay off!

CONCLUSION

Congratulations, the new application has been installed and everything is done, right? Wrong! At this point the users are using the new version but everyone should be alert to the fact that things could still go wrong.

We watch our first few live runs very closely. I review all the output to be sure counts look accurate, that the data itself looks correct on all the different reports. Are the codes that are being printed out valid values for each field? This is the type of error that is often overlooked because people assume that if there is data printed out on the report, that the data is correct. This is not always the case.

Users should be extra vigilant of any discrepancies in the new release. Whether it is a strange looking code, a VPLUS screen that doesn't look quite right anymore, or a report that perhaps doesn't total correctly. They should all be on the lookout for anything out of the ordinary. Even if the user turns out to be wrong about a possible bug, it is better that the problem be looked into as early as possible. We really encourage our users to come to us any time they feel that something is out of whack.

If problems do come up or bugs are found in the software, we have found that notifying the State immediately was to our advantage. In the beginning we attempted to fix them ourselves and then to notify them. Sometimes the fix we made was not written the same way as their fix and then we were out of sync, AGAIN. All bugs, whether potential or real should be noted on the checklist. Attach extra sheets to the back to keep specific notes on what the problem is, if it is a user error or a program bug, and what was done to solve the problem. On my checklist I have the heading "bug" and have three subheadings under it, "found", "reported to State" and "fixed".

Once everything has been finished - (you hope) - file your checklist and all of the attached extra sheets in a folder marked with the system name, version release and date of the final installation. For example, mine reads "CANDIS / ver.3.5 / 6-21-91". Many times people come back months later and wonder why something was done a certain way or who made a decision. It's now easy to pull out the folder and look it up.

We also realized that when we did call about a potential problem, we had to make sure that we did not call and start screaming. Funny how that immediately puts people on the defensive. It's far better to call and ask them to look into the problem or perhaps offer a possible solution (if asked). Keep the lines of communication open -- you may need them for something in the future!

A few final thoughts are:

- the importance of planning ahead
- the importance of not being rushed
- paying attention to the detail work
- carefully testing all programs
- using a checklist
- coordinating the install with the users
- keeping all documentation

Paper 3116

Software Re-engineering
Teaching an Old Dog New Tricks

Christopher Nelson
S/Wizardry, Ltd.
P.O. Box 869
Troy, NY 12180

Phone: (518) 273-5025
CompuServe: 70441,3321
Internet: nelsonc@cs.rpi.edu

INTRODUCTION

Modern business has come to rely on information systems. Many so-called "mission-critical" systems consist of custom software developed in-house or under contract. However well designed or documented these systems were when delivered, time marches on. Consultants go out of business, developers are promoted or leave the company or department, and generations of maintenance programmers leave their distinctive marks. Justifications and rationale die early in the life cycle of a system and its behavior becomes part of the oral tradition of the corporate culture.

Often, it is best to leave well enough alone but there are good reasons to modify entrenched systems. The first, and most obvious, is to correct a bug. Second, there may be a need to add new functionality. Finally, and a growing concern today, is to make the existing system a good neighbor in an open computing environment. This may involve making it network-compatible or network-aware, or porting to a new platform.

Over the past several years, I've worked on several projects that involved fixing, enhancing, or porting sizable software systems. I learned something new each time but I also benefited from my earlier experience in knowing what to do and what not to do. In this paper, I'll try to pass on some of my experience so that you may have a leg up if you are asked to take on one of these thankless jobs.

YOU CAN'T GET THERE FROM HERE

As an engineer for a large consumer goods company, I was involved in the design of a custom link between HP1000 supervisory process control systems and IBM System/38 business systems. When we began the project, we had asked both systems vendors and a number of third parties for their recommended solutions and failed to turn up even one system that fit the requirements. The closest we came was RJE/1000 II as a means of basic, bisynchronous communication to an IBM environment. However, the S/38 didn't support the host software for which RJE was designed and HP not only didn't support RJE with System/38s, they said it wouldn't work. Undaunted, we selected it as the basis for our systems and plowed ahead.

After designing protocols for application-to-application communication over RJE's bisync communication, I was moved on to other projects. An outside developer was contracted to develop the high level HP1000 software, and an in-house S/38 guru was found to write the IBM end of the link.

Several years later, I had moved from engineering systems to MIS and was a "fairly new guy" in a group supporting various technical applications. I had heard, from time to time, of the link I had designed: it was being used on a new process line, this or that problem had been found, etc. Then one day, I was asked to "look at a problem" they were having with one of the installations.

To make a long story short, the engineer on the project, the second or third in a couple of years, had a new assignment and fixing this "last little thing" was her last project. She didn't have design documents but had inferred some of the structure of the link. I, of course, had forgotten most of what I knew of the design and had never been involved in the implementation. Finally, the same guru who had worked on the initial implementation was a short timer (and sick of answering questions from engineers who didn't understand "his" system).

I spent some time with them trying to isolate the problem. We were able to set up test cases which consistently failed or succeeded but were not able to identify the significant difference between them. It became clear that we would not be able to fix the problem before their time was up. As a former engineer, I had worked with the HP1000 and as a member of the MIS staff, I was supposed to be able to work with the System/38. I ended up adopting the project and spending several intensive weeks trying to learn (or relearn) everything I could about the link.

The result of this crash course was a fresh knowledge of HP1000 communication, a vague understanding of System/38 operations, a list of contacts from various departments and vendors, a pile of diskettes and tapes of various sizes holding source code for both ends of the link, and tens of pages of hand-written and computer readable notes (some mine, and some passed on from the old guard).

From this pile, I began to synthesize an link handbook; something that made the scattered thoughts and notes coherent. I gained access to an HP1000 and a System/38 and had the appropriate software loaded on each. Then I used a terminal emulator to down load all of the source to my PC and merged it into the handbook.

To get a more global view, I gathered information on other installations of the link; what it was used for, how much data was transferred and how often, configuration details, and any problems encountered.

Through all of this, I kept a list of what I needed to know. Not surprisingly, the biggest gap in my knowledge was System/38 architecture and operations. After reviewing several IBM video tapes on general concepts, I arranged to take a week-long course on System/38 application development.

During the course, I kept my link problems in mind. When something in the course material struck a familiar chord, I'd stop the instructor during break and ask questions directly related to my situation. At the end of the week, I'd built a number of user defined commands and knew how to look more deeply into the structure of the System/38 link software.

In the end, I spent five months, full-time on the link. When I was done, I had solved not only the initial problem, but all the known problems at all the installations, and added a more usable interface on the System/38. I also produced a nearly 300 page handbook for the link which included introductory, user, and programmer information, a list of contacts, and full source listings.

Lessons Learned

1) The 80/20 rule of system documentation

Each time a system is handed off, it goes with 80% of the documentation which conveys 20% of the necessary information. After several hand-offs, what documentation there is, is useless and you just have to roll up your sleeves and investigate.

2) You don't have to know everything

Analytical skills are more important than system-specific expertise. A good analyst is more likely to find a bug than a developer who is expert in the system but has weak analytical skills.

3) Sometimes, one head is better than two

This is a corollary to the mythical man-month precept that adding manpower to a late project makes it later and is especially true of debugging and re-engineering. Even two people cannot hope to share information as well as parts of one brain.

4) Good things take time

Five man-months may seem a long time to fix one bug, and it is, but much more than one fixed bug came out of my efforts. Versions of the link which had been fixed and patched for several different installations were unified and the system as a whole was more usable and reliable.

5) Documentation is a good thing

To have spent as much time as I did reconstructing the system and not document it would have been irresponsible. The handbook I produced allowed others to take up the system after me without months-long investigations.

6) Give it to the "new guy"

A fresh recruit out of college doesn't, generally, have enough practical experience to tackle a re-engineering effort but a "first bounce", new from another department or company, is an excellent resource. First, he or she doesn't share your biases and will question more assumptions than someone who has been around a while. Second, asking around to fit together their puzzle is a good way for them to get to know who's who.

7) A PC is an invaluable tool

Especially for multi-host projects like this one, a PC makes a good focal point. Terminal emulation products are available for most hosts so multiple terminals are not necessary and source code can be collected for inclusion in documentation. Some very good analysis tools (including non-traditional tools like free text databases) are also more readily available for PCs than minicomputer and mainframe hosts. (If you are more comfortable with a workstation, many tools, though not all the same ones, are available there, too.)

Christopher Nelson

ZERO, ONE, MANY.

A year or so after finishing the re-engineering of the HP1000-System/38 link, I was asked to work on an application that used the link to communicate schedule and production information. The application had originally been written to communicate directly between one production line and the plant business system. Now a second line was being added and the application had to be adapted to support it.

I took some time to reacquaint myself with the link and acquaint myself with the application. As I examined the source code I realized that the original programmers were not as fussy as I was about comments or structure.

I eventually got to one subroutine which was the core of the application; cycling through reading from a buffer and writing to the process controller. This one routine spanned several pages and had a half a dozen or more GOTOs. It was hard enough to figure out what the code did; it was almost impossible to add support for a second line with any confidence that it would work.

I spent a full day untangling the spaghetti in the core routine. I replace all the GOTOs with IF/THEN/ELSE and DO structures and left one point of entry and one point of exit. At various points in the process, I stopped and tested the system to make sure it still worked correctly with one line. When I was done, it was about 20 minutes work to make some variables into arrays and add a loop to repeat the basic communications cycle for each of two or more lines; all that was needed to add more lines in the future was to change the constant used to dimension arrays and control the number of iterations of the outermost loop.

Lessons Learned

1) Structure is important

Some programming decisions are a matter of style but many are not. Using GOTOs is nearly as bad as some would have you believe. In this case it increased the time needed to modify the core communication routine by 2400% (one day vs. 20 minutes)!

2) Progressive refinement works

It is hard to digest a whole system at once. It takes time to get to know a system and there is no substitute for reading the code. By taking time to comment and "clean up" code as you go, you get some productive use out of the learning time. Also, there is a good chance that during or after the clean up, the solution to the problem will become obvious.

3) Two = Infinity

The original communication routine was written for exactly one production line. The code worked as intended but the lack of structure and other limitations inherent in the one-line assumption made it difficult to add lines to the system. On the other hand, once a clean loop (executed once for each line) was in place, adding a second or subsequent line was very easy. Build your solutions so that they work with "two" (documents in an editor, ports for communication, etc.) and you will get systems that are easier to understand and easier to expand or enhance.

A MUDDLED MASS

I was far from an expert C programmer when I took on revisions to a crystallography system consisting of 30 thousand lines of middling quality C code. Before I was through, I knew more than I cared to about C and the strange and wonderful programming styles it can foster.

The system included four programs to collect, edit, analyze, and plot x-ray crystallography data. The programs communicated mostly via files but, with one exception, the file formats were undocumented. The programs shared some common library code but duplicated other routines with common names and similar functionality with no apparent reason and no clear revision histories to explain the differences.

When I took over the maintenance of the system, it had been worked on by two other programmers over the course of three or four years. The two programmers had somewhat different styles and at least one of them was, in my opinion, rather sloppy and inconsistent. I set out to understand the system and to make the source code more readable.

First, I made a complete back up of the system as it existed when I got it. If I changed anything for the worse, I could at least get back where I started.

Next, I went through each of the major modules with a fine toothed comb, reformatting to my taste as I went, and adding or editing comments to clarify the function of small blocks of code.

Finally, after I edited each module, I compiled it. I turned on all the warnings the compiler had and went through the module again trying to eliminate all the warnings I could.

When I was done, I understood the overall structure of the system and began to see where things could be cleaned up at a higher level. I worked on identifying the most recent version of each duplicated library routine and, where appropriate, removing the older versions. The libraries also suffered from a lack of naming conventions for variables and routines so I set out to unify the naming scheme. (This was made easier by the lack of documentation since I didn't have to worry about updating it to reflect the new names.)

With the applications successfully rebuilt with simplified, clarified source, I got some test files to verify their operation. I got a quick lesson in the use of the programs and went through most menu options making sure that nothing blew up or had other undesirable effects. All seemed fine so I proceeded to adding some of the requested new features.

As I added features, I got into a routine of testing what I'd done and running through a few of the existing functions to make sure everything worked as it had. When I turned the first copy over to the users, I ran into some unexpected problems. It turned out that my use of the program during testing was particularly atypical. The combination and order of operations under normal use was so different that the users broke the program very quickly. I took copious notes and used a more realistic testing scenario for future revisions.

When the first pass of revisions to the existing programs was done, it was time to begin work on a new program that was to combine functions from two of the existing systems. I made an attempt to cut one down and add parts of the other but ran into some complications.

The first problem was the conflict of styles from the previous programmers. While I had unified a lot of names, calling conventions and the use of global variables made combining code from different programs difficult.

A worse problem was that the library organization was more web-like than hierarchical; routines in library A might call routines in library B but other routines in library B called routines in library A and both libraries might call routines in C.

I made an attempt at untangling the respective programs and their libraries before trying to merge them again. I got cleaner code but not nearly clean enough to merge them well. I put the problem "on the back burner" and began working on an outline for the new program, starting from scratch and building a new skeleton but hanging bits from the existing programs on the new bones. To my surprise, I soon had a program that looked a lot like the one I'd started out to make by merging the others.

The one omission in the prototype was an important one: one of the existing programs produced plots of the edited x-ray data using some home-grown plotting routines. They worked but added a great deal of complexity (and a lot of the problems that kept me from merging the programs).

As it turned out, the screen display was handled with a device-independent graphics library from the compiler vendor. The original library supported only display devices but a third party had produced a set of hard copy drivers which used the same API. Ideally, I could use the commercial library for my hard copy needs in the new program and end up with simpler code and support for more devices. To make a good thing better, the library cost less than a day's labor. There would be work to integrate it but less work than would be necessary to use the existing, home-grown solution. Suddenly, I went from a having a quick prototype missing an important feature to being almost done.

Lessons Learned

1) Have it your way

I said it before but there is no substitute for close examination of source code. Reformatting code and renaming variables and routines to your style not only makes it easier for you to work with but, if multiple programmers have left their marks before you, it leaves a more homogenous and more easily understood system for those that follow you. They may not like your style but at least it will be consistent.

Note: Don't bother with source beautifiers. They may format the code the way you want it but you won't get anything out of the process.

2) Know your limits

I used to wonder how Beethoven and others kept entire symphonies in their heads, writing the score for one instrument while somehow hearing how it fit into the whole. Now I understand. With years of practice, I can now hold the general form of 20-30 thousand lines of code in my head at once.

Another rule of thumb I use is that I can examine and reformat code at a rate of approximately 50 kilobytes a day, two to three days a week. More code than that and I'm working mechanically and not getting anything out of the process. More days than that and I risk injuring my eyes or wrists.

3) Test procedures, as well as data

I thought I was doing well by getting existing data files to test the new and revised programs. Clearly, that was not good enough. If you don't know exactly how the programs you are working on will be used, get a written test plan or get users involved in the testing process. The test plan doesn't have to be anything grand and glorious, just a few one-page scripts describing the procedures to go through to run typical functions of the system.

4) Starting over is not a bad thing

I might still be trying to merge the two programs if I hadn't decided to take some time off and prototype. If you need to build something like something else, learn all you can about the original then decide if it is better to start over, borrowing small chunks as appropriate, or to adapt the existing system. The construction of well built systems is dominated by analysis so don't be afraid to start coding from scratch.

5) Don't re-invent the wheel

Programming is expensive and error-prone. If you can buy the technology you need, do so. Especially in the PC arena, many programming tools and support libraries are available in source form so you can reasonably protect yourself from a vendor going out of business. In the final analysis, you are no worse off having to figure out their code than something written by the guy down the hall.

A TALE OF TWO SYSTEMS

What would you do if you were asked to port a process analysis system from VAX Pascal to generic C? You might laugh and leave but I wasn't quite that smart. I took on the project with some interesting results.

The original process analysis system was written in Pascal using a terminal-based user interface system for the VAX. It collected some data automatically and prompted the user for some additional values then calculated efficiencies related to running some processing equipment.

Before I got the project, someone had manually cut out the user interface code and used a porting tool to turn the resulting, rather vanilla Pascal into C. When compiled, this C code produced an executable program about a half a megabyte in size that seemed to produce good results but had not been rigorously tested. My assignment was to "clean up" the auto-ported code and then develop similar modules for down-stream process gear.

As I had with the crystallography programs, I began by making local changes to the major modules and trying to get a feel for how the whole thing fit together. I soon realized that the code was very poor; it was, as you might expect, Pascal structures in C syntax. My earlier success with prototyping suggested that I try it here to get a clean C implementation of the same functions. Again, I built a new skeleton and moved bits of the old program onto it. In the end, I had a cleaner program that was one seventh the size of the auto-ported version and produced the same results! I also had good, clean templates for constructing the new modules and their development proceeded very quickly.

An added constraint for the new programs was that they were to be portable. Minimally, they were to run under MS-DOS and Xenix. To enhance portability, I told the compiler to recognize only ANSI standard C and to give me all the other warnings it could. In some cases, it took a couple of revisions to get things working under this restrictive scenario but, eventually, it all worked.

I had been working with Borland C but found that one of the features of the new programs could only be implemented with a library of routines compiled for Microsoft C. My efforts to be standard and portable made me confident that this would not present a problem: just rebuild all of my code under Microsoft C. Alas, Microsoft's idea of ANSI C and Borland's differ. I don't have the standard so I don't know which, if either, is right but there were numerous discrepancies; nothing serious but discrepancies nonetheless. In the end, I was able to resolve all the problems with only one line of code dependent on which compiler was used. Everything else could be done in a way that both compilers accepted. The system works and the four modules taken together are still smaller than the original auto-ported version of the first module.

Lessons Learned

1) The best optimizer is between your ears

The auto-porting program saved a lot of trouble and error-prone work of converting Pascal to C. Still, it was a syntactic conversion. The language paradigms were not bridged and the resulting semi-C program was not very efficient. By using it as a model, a much more efficient version was implemented with fair ease. The auto-ported program also served an important function in giving a "gold standard" to which the new program's output could be compared.

2) Two compilers are better than one

No matter how careful you are and how good your compiler, every programmer and every vendor has quirks. It would be ideal to have another programmer examine your code in some detail to find your eccentricities but it is much more practical to have a second compiler on hand. A third is not overkill if you want solid, portable code.

CONCLUSION

Software re-engineering is not easy. It requires good tools, good analytical skills, and determination. However, it is often the only way to protect or recoup an investment in poorly understood or poorly implemented legacy systems. Properly done, it can lead to more open, more portable, and more functional systems which are easier to maintain than their predecessors.

Object-oriented methodology: the Adager instance

F. Alfredo Rego

Adager Corporation
Sun Valley, Idaho 83353-0030, USA

Telephone +1 (208) 726-9100, Fax +1 (208) 726-8191

WE HEAR A LOT ABOUT object-oriented programming (OOP) using object-oriented languages (OOL) and object-oriented database management systems (ODBMS) in an environment guided by object-oriented analysis (OOA).

The fundamental concepts are simple and powerful, but there are two barriers around the object-oriented territory: the *syntax* of the sundry implementations and the *generic jargon.*

I do not discuss the syntax of the motley object-oriented implementations, because such syntax is easily available through their accompanying technical manuals.

I address the generic jargon, which can be intimidating: *class instance, superclass, method, message, encapsulation, information hiding, overloading, inheritance, overriding, polymorphism, trigger,* and so on.

To illustrate object-oriented principles I employ the database concepts that I use with ***Adager***, The ***Ada***pter/Mana***ger*** for IMAGE/3000 Databases (Turbo, XL, or iX). I look at things from an IMAGE perspective, weaving—perhaps several times, for reinforcement—through a set of definitions as a way to explore the qualities of the object-oriented landscape.

Where do you begin?

THERE ARE COUNTLESS views on the object-oriented issue. Let's look at some examples.

Programmers think about **object-oriented software**. Not only that, programmers who cut their teeth on C swear by *either* C++ *or* Objective-C (but usually not by both); those from the Pascal camp defend either Turbo Pascal or Object Pascal. Following behind C and Pascal, all major programming languages (including good ol' Lisp, Cobol & Fortran) are adding object-oriented extensions to their repertoires. Meanwhile, the purists—best represented by the various Smalltalk contingents—decry the impurities of *all* such hybrid languages:

> *The Smalltalk and C++ camps represent two distinct programming cultures. The "bitmap, window, and mouse" culture represented by Smalltalk is part of the user-interface revolution that began in the 1970s. On the other hand, the "single-character-prompt, save-memory-and-machine-cycles-if-it-kills-you" culture is represented by C++.*[1]

Recently, BASIC has made a strong comeback with VISUAL BASIC.[2]

Tom Keffer is careful to point out that *"all object-oriented code can be written in assembler, but that does not make assembler itself object-oriented.*[3]*"*

1. Rettig, Morgan, Jacobs, and Wimberly (1989). *Object-oriented programming in AI: New choices.* Article in *AI Expert,* January 1989.

2. *"Of course,"* says Wirt Atmar. *"Boring, boring,"* says Berni Reiter. *"Astonishingly,"* says Tom Keffer. (Private communications).

3. Private communication.

Then, what does "being object-oriented" mean? Does it mean to have "built-in" capabilities that automatically take care of the object-oriented stuff for us? Does it mean to *encapsulate* "information" with "the intelligence required to process that information"? Does it mean to be able to define your own data types in addition to canned—primitive—data types?

Do you need to deal with "complex and varied" things to be a candidate for the object-oriented life? Do you have to use multimedia—such as video or audio—in your database?

There are so many views on this issue. Let's look at two contributions which might appear unrelated to the object-oriented hype, as examples of the wide spectrum of possibilities.

Donald Knuth, implicitly, brings fundamental object-oriented ideas to the very act of *literate* programming:

> *Literate Programming is a programming methodology that combines a programming language with a documentation language, making programs more robust, more portable, and more easily maintained than programs written only in a high-level language.*
>
> *Computer programmers already know both kinds of languages; they need only learn a few conventions about alternating between languages to create programs that are works of literature. A literate programmer is an essayist who writes programs for humans to understand, instead of primarily writing instructions for machines to follow. When programs are written in the recommended style*

they can be transformed into documents by a document compiler and into efficient code by an algebraic compiler.[1]

There are researchers in Artificial Intelligence and Robotics who investigate the concept of *autonomous agents*, and who concern themselves with:

...the radically different architectures that have been developed over the past few years for organizing robots...

These architectures emphasize more direct coupling of sensing to action, distributedness and decentralizing, dynamic interaction with the environment, and intrinsic mechanisms to cope with limited resources and incomplete knowledge.[2]

Besides programmers, there are *"databasers"* who think about **object-oriented database management systems**. People who work with databases think routinely in terms of **persistent objects**—a notion that seems to be a novelty among people who deal with programming languages.

Among database people, there are those who believe in adding object-oriented extensions to traditional **D**ata**B**ase **M**anagement **S**ystems (DBMS)[3] and there are purists who must build object-oriented database management systems (ODBMS) absolutely from scratch.

1. Knuth (1992). *Literate Programming*. Center for the Study of Language and Information, Stanford University.

2. Maes (1990). *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. The MIT Press.

3. Hewlett-Packard has chosen this approach for OpenODB.

Adager

ADAGER is a software system with one mission: to manage the structural elements of Hewlett-Packard's award-winning IMAGE database management system. Consequently, Adager knows all the structural rules that govern IMAGE databases. In addition, Adager knows its own internal state at any given time as well as the state of the database it is managing. Whenever you send a message asking Adager to perform a given transformation on a given IMAGE database, it decides—at run time—whether the requested behavior is appropriate.

The suitability of your request is totally unpredictable. Adager *must* postpone its operational decisions until it has all the required information. Adager must delay, as much as possible, the process of binding its methods to your messages.

During my years designing and building Adager, I have learned quite a few tricks of the trade that I would like to share with you in the context of this object-oriented essay.

Dizzy?

DON'T FEEL BAD if you become dizzy, since you are in good company—just watch the sundry committees trying to impose *their* respective standards on the rest of the computer community, usually with total disregard for the suitability of their edicts.

If you think that programming languages and database management systems are a *mêlée*, you have to brace yourself for the *standards* gyrations.

Suitability

SOMETIMES, "the best"—according to a given set of someone else's criteria—is not necessarily the most *suitable* under a given set of circumstances for you.

In this article, I will stress the fact that, besides absolute decrees from standards committees, we must always consider how appropriate—or inappropriate—a proposed solution is for a given purpose or occasion.

A definitional tour of the object-oriented landscape

SINCE CHILDHOOD, I have enjoyed playing with definitions. I have always believed that *pursuing* clear and distinct definitions—even as we get frustrated by their elusiveness—is a fundamental prerequisite for understanding any subject. The act of tossing buzzwords about *without* having a clue regarding their meaning can be dangerous and we should avoid it.

So, in this paper, I explore a set of definitions that describe the basic qualities of the object-oriented territory.

For the sake of reinforcement, I may visit some definitions more than once. Hopefully, each additional visit will, in a spiral-like fashion, bring us closer toward our final goals: awareness, appreciation, enjoyment.

Entities and relationships

WE MUST BEGIN somewhere. Because I spend a lot of time dealing with databases, we might as well start our exploration using database terminology.

A DBMS keeps track of **entities** and their "strategic alliances," or "involvements," or "connections," or "entanglements," or "embroilments," or whatever you want to call them—which we shall name **relationships**.^{1}

Attributes

THERE ARE **specific entities** (for instance, the employee whose name is *Jan López*, or the company called *Control Engineering*, or the department called *Sales*). Each specific entity has its own specific **entity attributes** (such as Jan's employee number, or her date of birth).

There are also **specific relationships** (for example, the *assignment* relationship between the entity called *Jan López*—an employee—and the entity called *Sales*—a department within the company). Each specific relationship has its own specific **relationship attributes** (such as the starting date of Jan's *assignment* to the Sales department, which may be different from the starting date of Jan's *employment* at Control Engineering).

Keys

FOR ANY given entity or relationship, I choose a set of attributes as a **key** that *uniquely* identifies it. The key is composed of one or more attributes that form a unique *handle*. (Wirt Atmar prefers to think of the key as the *noun*, separate and independent from the attributes—which he considers as *adjectives*. I prefer to think of the key as a subset of the attributes.)[1]

Intelligent attributes

WE MAY CERTAINLY think of attributes as just *pieces of data*, but I prefer to think of attributes as a com-

1. We must also keep in mind the meanings of terms such as *relation* (used in Dr. Codd's *Relational Model*) and *relationship* (used in Dr. Chen's *Entity-Relationship Approach*).

1. We are still friends.

bination of **data** <u>and</u> the **intelligence** required to process the data.

As an instance, let's study our treatment, in Adager, of the *name* attribute as we apply it to the names of individuals in our customer database. We have implemented this technology easily, within the rules of IMAGE. We keep the *information* for names thus:

<LastName><Separator&Title><FirstName>.

LastName and *FirstName* may have only *one* name ("Smith," as in the U.S. tradition)—or *many* last names ("Montes Fernández de García Salas," as in the Guatemalan tradition). *Separator&Title* is one of several codes (for instance, ":" for "Mr." or ";" for "Mrs."). *Separator&Title* serves a triple function: (1) as a separator between *LastName* and *FirstName*, (2) as a clue to the person's "title" or "greeting qualifier" and (3) as a data compressor, since there is no need for a blank to separate *last names* from *first names.* My name attribute, for instance, is encoded as "Rego;Francisco Alfredo" and decoded as "Mr. Francisco Alfredo Rego" (or decoded as "Mr. Rego" for a greeting).

In addition to the *information*, we keep the *intelligence required to process this information.* There are two actions that we must perform on names: (1) To *encode* them from the human way *(Mr. Francisco Alfredo Rego)* to our internal Adager way *(Rego;Francisco Alfredo)*, and (2) to *decode* them from our internal way back to the human way.

As we shall see under *Methods* and under *Objects* below, the Adager treatment of attributes—as a combination of *information* **and** *the intelligence* *to process that information*—is a necessary condition in object-oriented thinking.

Data types

THE CONCEPT of *data type* is fundamental in database management. It is express as the idea of *domain* in the relational model.

There are *built-in* or *primitive* data types and there are *user-defined* data types. For instance, in IMAGE, there are these primitive data types: **I** (integer, from 16 bit through 32 bit to 48 bit and higher); **J** (same as I, but digit-oriented as in Cobol instead of bit-oriented); **P** (packed decimal, from 3 to 27 digits); **Z** (zoned decimal); **K** (unsigned integer, logical, or boolean); **X** (unrestricted character); **U** (uppercase character); **R** (HP3000 real, floating point); **E** (IEEE real, floating point); etc. In addition to IMAGE's primitive data types, Adager supports two user-defined data types (introduced by AQ/3000 more than a decade ago) for *date (D)* and for *monetary (M)* information. There is no limit to the number of user-defined data types that IMAGE can support.

For instance, Wirt Atmar[1] is very enthusiastic about cooperating with HP in defining a BLOB (**Bi**nary **L**arge **Ob**ject) data type within IMAGE. Most standard BLOB implementations concern themselves with bitmapped, rasterized images. Wirt's vision goes much beyond that: he suggests using PostScript as the language of choice for describing the intricacies of the kinds of objects that we want to store by means of the BLOB data type.

As another instance, consider my use, within the Adager Corporation, of IMAGE databases to sup-

1. Private communication.

port object-oriented implementations such as the *name* example we just discussed under *Intelligent Attributes.*[1]

Classes

IT IS FINE to think in terms of **specific entities** and **specific relationships**. But after a while, it is convenient to go up one level in the abstraction ladder.

In our case, we introduce the concept of **class** and we apply it to entities as well as to relationships.

An **entity class** is a set of entities whose members have at least one *entity attribute* in common (for instance, *company employees* is an entity class that contains all the employees in the company).

A **relationship class** is a set of relationships whose members have at least one *relationship attribute* in common (for instance, *company employment* is a relationship class that contains all the employment relationships in the company).

In practice, we usually refer to an *attribute* or to a combination of attributes. A standard way to refer to a fully-qualified attribute is to use a combination of *class name* <u>and</u> *attribute name.* For instance, it makes sense to say *employee salary* (where *employee* is the class name that includes all employees and *salary* is the attribute name). Therefore, to avoid awkward results when building fully-qualified names for attributes, we should assign *singular* class names (such as *employee*) as opposed to *plural* class names (such as *company employees*).

1. If we can do this with IMAGE now, given its current limitations, just imagine what can be done as Hewlett-Packard continues its development of its award-winning IMAGE database management system.

General relationships

WE DO NOT HAVE TO restrict relationships unnecessarily. In fact, most relationships should allow a given entity to be related to zero, one, or more entities. Under special circumstances, we may want to apply restrictions—but restrictions should *not* be the default. Let's take *marriage* as an example. Under strict Catholic rules, a priest or a nun should be married to *zero* people. Under currently-standard law, a person can be married to *one* other person—of the opposite sex in most jurisdictions, but not necessarily so in others. Under Mormon rules a century ago, a man could be married to *many* women. Regardless of your moral—or legal—restrictions regarding marriage or any other relationship, your DBMS should *not* force its restrictions on you

The *number* of entities involved in a relationship is just one dimension. The *kind* of entities involved in a relationship is another dimension. The involved entities may be members of different classes—for instance, an employee may be related to a department through an *assignment* relationship. Or the entities involved in a relationship may be members of the same class—for example, an employee may be related to another employee through a *marriage* relationship.

The Relational Model

UNDER THE Relational Model, I like to represent **entity classes** and **relationship classes** as *relations*, and I like to represent **specific entities** and **specific relationships** as *tuples.*

(A *table* is one possible—but not the only—representation of a relation. A *row within a table* is one possible—but not the only—representation of a tuple. Many people confuse these terms, so you want to be careful.)

Transactions

To be able to keep track of entities and their relationships, a DBMS has to "keep books" and, just like a bookkeeper, a DBMS juggles **transactions**[1] that *add, modify* and *delete* **entries.**[2]

IMAGE

Under IMAGE, I like to represent **entity classes** and **relationship classes** by means of *datasets*, and I like to represent **entities** and **relationships** by means of *entries.*

If performance is not a problem, we can always keep our information in simple tabular form. But if performance is an issue—particularly if you have millions of entries—you may want to take advantage of smart database structures, such as those of IMAGE.

From a performance standpoint, IMAGE **master datasets** are optimized for fast hashed access which allows us to quickly find a given entity (master entry) among millions. IMAGE **detail datasets** are optimized for immediate access to all relationships (detail entries) associated with a given entity (master entry).

With IMAGE, you have the freedom to hard-wire well-known, repetitive, standard, "hot" relationships by means of *paths*, if you desire, to dramatically increase the performance of your routine database searches. (But, naturally, you do *not* have to do so if you do not desire. You can always bushwhack with your machete through an unchartered

1. *Transaction* comes from the Latin *transactus*, past participle of *transigere*, which means *to complete.*

2. I find the term *entry (an item inserted in a diary, register or reference book)* to be more user-friendly to non-mathematicians than the term *row (in a table or matrix).*

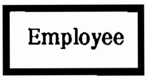
jungle every time you want to look for an entity or for a relationship.)

Adager allows you to add and to drop IMAGE paths very quickly, but it is important to realize that you do *not* have to use IMAGE paths at all. IMAGE's paths (and their pointers) are nonessential and do not convey any *information about your information*—as is the case with standard hierarchical and network database management systems.

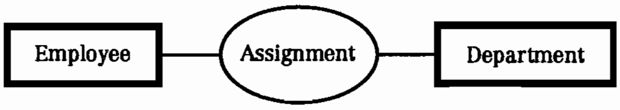
Diagrams

THROUGHOUT the years, I have developed a way to diagram entities and relationships. I can easily translate these diagrams into standard relational database designs and, of course, into IMAGE database designs—since IMAGE is closer in spirit to the Relational Model than most people realize.

To diagram **entity** classes, I like to use **rectangles**—perhaps because entities tend to be somewhat stable and rectangles convey a feeling of steadiness. For example, the **entity class** *Employee*, which contains all employees in a company, gets a diagram like this:



To diagram **relationship** classes, I like to use **ovals with outstretched lines**, reaching out to touch the rectangles that represent entity classes—maybe because they remind me of nice people with outstretched arms, reaching out to touch others. For example, the **relationship class** *Assignment*, which contains all employment assignments between the class of employees of a company and the company's class of departments, gets a diagram like this:



The literature related to *Graph Theory*—and to the *Prolog* programming language—tends to use *circles* for "objects" and *connecting lines* for "relationships between objects." You may consider my *ovals with outstretched lines* as "lines that happen to have a *lump* in the line" (just for the convenience of being able to write the *name* of the relationship in the lump/oval).

Normalizing

IN THE LAST DECADE, I have developed a wry[1] working definition of normalizing: *Keep together those things that belong together and separate those things that do not belong together.*

Deciding which things belong together is, obviously, a matter of taste. Since we have all been exposed to standardized taste samples in the database literature, I would like to treat you to a refreshing new taste.

Here is an example that deals with two kinds of entities—*employees* and *departments*—and with two kinds of relationship between them—*assignment* and *management*. In many books on database

1. My dictionary defines *wry* as *dryly humorous, often with a touch of irony.*

management systems, we see this classic example treated along these lines:

Employee relation (represented by a table)

Emp#	EmpName	Dept	Salary	Mgr
123	Jan López	Sales	55	Jane Blow
235	Pete Rabbit	Mktg	90	Sue Plus
813	Max Holz	Design	25	Fritz Kuh

This standard treatment is fine from a performance viewpoint, since a given entry—or record, or row, or "get"—brings everything you want to know about a given employee. But I believe this approach crams too much into the *Employee relation,* thereby obscuring the model.

Performance vs. modeling power

PERFORMANCE is, most certainly, a worthy goal. Modeling power is another worthy goal. Sometimes, unfortunately, these two praiseworthy objectives are at odds and we must make thoughtful trade-offs.[1]

In this example, I want to emphasize that a good DBMS should allow us to model anything we want, without forcing a "standard" framework on us. Since most of the database literature explains the highly-unnormalized approach illustrated in the previous table, I would like—for balance's sake—to illustrate the other end of the normalization spectrum. Any point along this wide range of

1. A *trade-off* is *an exchange in which something desirable, as a benefit or advantage, is given up for another regarded as more desirable.* The crux is in the word "desirable," since one person's junk is another person's treasure. Standards committees go insane over this issue.

normalization choices is perfectly acceptable, as long as we know *why* we are selecting it.

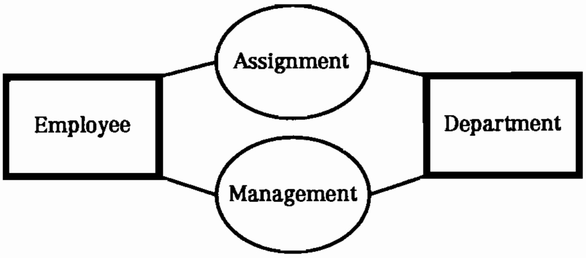
High degree of normalization

We can normalize this relation by *separating* its attributes into *four* distinct relations: (1) attributes of **employee entities**—such as *Emp#* and *Emp-Name*; (2) attributes of **assignment relationships**—such as *Emp#, Salary,* and *Dept*; (3) attributes of **department entities**—such as *Dept*; and (4) attributes of **management relationships**—such as manager's *Emp#* and department's *Dept.*

By doing this, we lose something and we gain something—as is the case with matters of taste *and* with matters of economics. **The losses:** We use four relations instead of just one relation. Besides the proliferation of relations, we may cause more disc accesses—thereby lowering the performance of our database accesses. **The gains:** We have a more flexible model of reality and we do not need to introduce the concept of *nulls*[1] at all. We allow employees to be assigned to zero, one, or more departments—with different salaries for each assignment, including multiple assignments to the same department. We allow departments to have zero, one, or more managers. We allow a given employee to be a subordinate in some department(s) and a manager in some—presumably *other*—department(s), and so on.

1. *Nulls* are a cumbersome idea that some people have proposed to deal with information that is missing from the database. Some of the missing information may be *applicable* and some may be *inapplicable.* I have found that, by simply separating entity attributes from relationship attributes, the concept of nulls becomes unnecessary.

Here is a diagram:



This is a table—with a few other appropriate attributes—for the *Employee* ***entity class:***

Employee (Entity class)

Emp#	EmpName	BirthDate	BirthPlace
123	Jan López	19450926	USA
235	Pete Rabbit	19601203	UK
813	Max Holz	19540514	Perú

This is a table—with a few other appropriate attributes—for the *Department* ***entity class:***

Department (Entity class)

Dept	Budget	DateOfCharter
Sales	1000	19880213
Mktg	50000	19900203
Design	370	19770212

This is a table—with a few other appropriate attributes—for the *Assignment* ***relationship class:***

Assignment (Relationship class)

Emp#	Dept	Salary	StartDate
123	Sales	55	19920514
235	Mktg	90	19900405
813	Design	25	19910506

This is a table—with a few other appropriate attributes—for the *Management* ***relationship class:***

Management
(Relationship class)

MgrEmp#	Dept
4528	Sales
4321	R&D
7704	Support

Sneak preview

LATER ON, I shall use the same highly-normalized database modeling technique to diagram my synopsis of object-oriented thought. Before doing so, though, we should review some of the object-oriented definitions.

Behavior and messages

ENTITIES HAVE some kind of **behavior**: they must be able to **act**. Entities *affect* other entities and, in turn, *are affected by* other entities.

A basic thing that entities do is to communicate with one another by means of **messages**.[1] Some of

1. We may think of *message passing* as analogous to *calling a procedure with late binding.* And we may also think of message passing as analogous to *packet switching* in network & telecommunications terms.

these messages are "sweet nothings" and some are quite "serious." Naturally, one person's sweet nothings are another person's deadly-serious messages—and vice versa. In general, though, an entity that sends a message (the **sender**) hopes that the **receiver** will take some action according to the receiver's usual behavior. Ideally, the receiver will not just take any old action: there should be some **method** to this madness.

Greg Voss believes that *"object-oriented programming is programming by sending messages to objects of unknown type."*[1]

Objects

LET'S CONSIDER NOW a special kind of entity that (1) is capable of sending and receiving messages from other entities and (2) has predictable behaviors that depend on the meanings of the messages it receives and on its own internal **state**. Let's refer to these entities as **objects**.

Different people define *objects* in different ways. Take your pick from this small sample: Some say that *"objects are modules that contain both data and the instructions that operate upon those data,"* others prefer to think of objects as *"anything that can be named,"* and Smalltalk, as a "pure" object-oriented language, specifies that *everything* is an object.

Active objects

REGARDLESS OF our exact definition of what an object *is*, we can do a further classification of objects (just as we do with our friends). Some are *couch po-*

1. Voss (1991). *Object-Oriented Programming.* Osborne McGraw-Hill.

tatoes (**passive objects**) that do something only if explicitly requested. Some are *eager beavers* (**active objects** or **agents**) that run all over the place looking for trouble.

Object identity

OBJECTS, being a special kind of *entity*, inherit all the properties that entities have. In particular, they inherit the property of having a *key*—a *unique identifier* that distinguishes a given object from all other objects within a system.

In our object-oriented diagram, below, we will see that the *CompositeObject* relationship depends on unique object identities (object keys) to establish the "**B**ill **O**f **M**aterials" (BOM) that describes how a given object forms part of zero, one, or more **composite** objects, as well as how many **component** objects—zero, one, or more—form a given object.

As with any entity class—and relationship class— within a database, you can establish fast access paths for "hot" and "heavily accessed" objects. For instance, in IMAGE terms, you can define hashing and chaining if you so desire—and you can repack specific datasets to cluster kindred objects physically next to each other, to improve performance even more.

Methods

METHODS, in object-oriented thinking, are *attached* to **objects**. A method is a particular set of well-defined operations that can manipulate the object's internal *state* variables as well as its internal *data* variables. We cannot directly operate on objects. We must obey a strict protocol. If we want a given object to perform some task or operation—if we want some object to *behave* in a certain way—we must send it a polite message. We cannot bypass this convention.

Sender objects have the responsibility of choosing qualified receivers. In Spanish, we say, *"no hay que pedirle peras al olmo"* (roughly translated, this means that you should not expect an ASCII terminal to produce fractal images in color—any more than you should require an X terminal to cost just a couple of hundred Dollars).

A *method*, then, is something that an object can do for us—if we can muster up the appropriate incantation.

A *message* ***is*** such an incantation—and we hope that it is appropriate. Otherwise, we face a **bug**: the correct message to an unsuitable object or the wrong message to an otherwise tailor-made object.

Responsibility

THE BOTTOM LINE is that, if we want to have a stable edifice, we must rely on **responsible** components.

Whoever is calling the shots (the *sender*) is responsible for the *shots* as well as for *calling them.* Timothy Budd calls this *"Responsibility-driven design: A design technique that emphasizes the identification and division of responsibilities within a collection of independent agents."*[1]

State transitions

WE CAN SPEND our lives sending messages right and left whose only destiny is to meet with deaf ears.

How many times have you desperately hit the RETURN key (or the ENTER key, or a mouse button, or whatever interface your favorite computer uses)

1. Budd (1991). *Object-Oriented Programming.* Addison-Wesley.

to no avail, while your favorite (?) computer does its own thing, refusing to pay attention to you?[1]

What is going on? Usually, the computer is in a non-receptive state—due to a system crash, or to a very busy schedule whose priority system does not think highly of you (A program running amuck while the *break* key is disabled[2]? An infinite loop? A system hang?)

Just as *Methods* are one important component of objects, **state variables** are another vital constituent of objects, used to keep track of state transitions. An object can only respond *well* to your messages if it happens to be in an appropriate—receptive—state. Otherwise, the object either does not respond at all (if it is poorly designed) or responds with a "busy" message (which is annoying but at least you know what is going on).

1. This is rare in the HP3000 environment. It is quite common in other, run-of-the-mill computing environments. Doubtful? Just run your own survey by asking your friends.

2. Through a system manager's choice, or through the programmer's choice, or through your own choice—which you may regret.

We mentioned *sender* objects. If we backtrack enough, we will eventually find some **prime** mover: the originator of the whole thing.

Prime movers & triggers

We, as humans, initiate a whole chain of events by issuing the original request to the system, usually through a **graphical user interface**, commonly called a GUI.[1]

We stimulate the GUI into some behavior, which typically results in the GUI object passing a message to some other object. This continues along the chain of command until, eventually, **the rubber meets the road** and some lowly object(s) actually execute(s) the desired duty.[2] These underlings carry names such as *primitive*, *essential* or *fundamental* objects.[3] They may carry out their behavior either sequentially or in parallel, depending on the *orchestrating* abilities of the "higher" objects and on the "instruments" being "played" by the lowly objects. All in all, quite a management challenge.

As human representatives, some specialized (**trigger**) active objects can constantly examine—poll—the **state** variables of certain telltale objects. Whenever a trigger notices that a given state changes, it compares the state differences with

1. As an aside, note that most "graphical" user interfaces contain, besides pretty graphics, an important element of touch—such as manipulating a mouse to point and click. Every day brings new and exciting interface technologies based on other human senses, besides sight and touch. Voice recognition is available now. How about sweat-and-tears recognition? Would a GUI then become *gooey?*

2. By means of their *methods*, which are conditioned responses to a specific repertoire of *messages*.

3. Somebody has to create these primitive objects from scratch. Nowadays, new businesses have sprouted whose mission is to create *class libraries* for your convenience.

some rule (or with some stored table, or uses some algorithm) to determine whether a given action is required. If so, the trigger then decides *which* object is the most appropriate underling and sends it a message to initiate the appropriate chain of events.[1]

Interpretation of messages by objects: overloading & polymorphism

THERE ARE two special circumstances under which we can fine-tune the interpretation of messages.

In the first case, **different objects** (or the same object under different circumstances) can interpret the same message in different ways (**overloading**). For instance, if you say "go," a runner will begin to run, a bicyclist will begin to pedal, a hot-air balloonist will go up, a scuba diver will go down. And there is nothing to forbid the same person from engaging in all of these athletic endeavors—presumably at different times. If you are the start-referee, this is quite convenient for you, since you have to learn only one generic message—*go*—that, according to the circumstances of the receiver, will generate a wide spectrum of actions. Naturally, you are still responsible for selecting your inventory of receivers—and their circumstances—in a reasonable way.

Another example, brought to my attention by Wirt Atmar, is the hormonal mechanism in living organisms. A hormone, as a *message*, is a very simple molecule that gets "thrown into" the blood

1. An alternative approach is the model-view-controller object, which uses an interrupt-driven method rather than a polling method. This is widely used in *window* types of user-interface applications.

stream (or the sap stream) by some *sender*. The *receptor* reacts to the hormone according to a combination of the *hormone* itself plus the receptor's own *nature* and *state*. Adrenaline, for example, causes some organs to speed up and other organs to slow down.[1]

In software terms, the receiver can have hard-coded (or hard-wired) ways to respond to your messages if you want to be superefficient. But, if you prefer to be flexible and to be free to pursue the best course of action at any time in the future, you may want to **bind** as late as possible, taking advantage of the **overloading** property. In the Adager *name* example, for instance, I use MPE's *LoadProc* intrinsic to invoke, at run time, the appropriate *procedure*—the appropriate method—according to up-to-the-nanosecond values of internal state & data variables.

In the second case, **the same object** can act differently—can invoke different methods—depending on the **mode**(s) of **polymorphic parameters**. In *monomorphic* systems, there is a **unique type** associated with procedures and with their operands. For instance, in a strongly-typed language such as SPL, you cannot add a *logical* variable to a *byte* variable without explicitly "converting" one to the other.[2] In *polymorphic* systems, a given parameter

1. Wirt says that the laws of Informational Physics were discovered several million years ago by living organisms. I believe that, instead of using the bland term *object-oriented*, we might want to use the nicer term *organically-oriented*.

2. SPL gives you the *check* option, by means of which you can specify how much parameter checking the segmenter will do on your behalf at "link" time.

name may refer to variables or objects whose types may be of several different types. For example, the parameter name *number*—in the abstract—may be "instantiated"—in the concrete—by an integer number in one case of addition, by a real number in another case, by a zoned-decimal number in still another case, and so on. These specific kinds of numbers are subclasses of the more general "number superclass." So, subclasses can share code designed with the generic superclass in mind. In the athletic example above, all people responding to the "go" message were members of the superclass of *athletes* and knew how to interpret the *go* message according to their specific *sport* subclass.

Nowadays, people tend to use *overloading* and *polymorphism* to refer—more or less—to the same concept: the avoidance of the *case* statement,[1] since each object knows its own type. By knowing its own type, each object is able to bind the appropriate *method* as late as possible to a given *message*. There is no need to bind at compilation time with a bunch of *if* statements that, to be extended in the future, would need recompilation.

Organizing objects

A BUNCH OF OBJECTS can become quite unruly unless we organize them. There are two main ways to organize objects.

The first way groups objects in a **compositional** manner which is similar to a bill of materials. For instance, we may think of an animal as having a head, a trunk and extremities; the head, in turn,

1. A *case statement* is a disguised series of *if* statements. In the *numbers* example above, a *case* statement would say, "if integer, do integer addition; else, if real, do real addition..."

may have eyes, ears, nose, mouth, etc. A **composite object** consists of other objects which we call its **component objects**. (Whether or not the component objects are "simple" and the composite objects are "complex" is really not an issue for me. I prefer to think in terms of a given object **being composed of** zero, one, or more objects—and in terms of a given object **forming part of** zero, one or more objects.)

The second way organizes objects according to an **abstraction** of their properties. We saw that the two main properties of an object are (1) its method(s) and (2) its internal—state and data—variable(s). It makes sense to abstract (or *factor out,* if you will) as many properties as possible at any given level and to keep track of them at the next-higher level.

From any given level of abstraction, all lower levels **inherit** that level's properties. For instance, instead of keeping information on "knows how to drive on the *right* side of the road" for every (properly) licensed driver in Guatemala, we can record this information in the **class** of Guatemalan drivers. Similarly for British-tradition drivers, who know how to drive on the *left* side of the road.

A **class instance** is a specific object that is a member of a class. For example, René Woc is an instance of "Guatemalan driver" and Bob O'Brien is an instance of "British-tradition driver (with an Australian bent)."

Overriding

FORTUNATELY, class instances are not condemned to blindly follow the dictates of their class. A class instance—or a subclass—can **override** a class *method* or a class *state variable* with a more appropriate one (during the act of binding the selected method to the given message).

Class hierarchies and inheritance

Classes, besides consolidating objects, can be consolidated themselves into groups of classes, forming **class hierarchies**.[1]

For a given class in a class hierarchy, we call those classes "below" it **subclasses**. Subclasses inherit all the properties of the given class but have, in addition, properties of their own which are *not* properties of the given class. As we go "down" in the class hierarchy, we have more differentiation or specialization—while still keeping the fundamental properties of the given class. For instance, in IMAGE we may have (a) a *generic* "data item" which, when defined in a dataset, becomes (b) a "field" which, if you choose to optimize access to entries with specific values defined by that field by means of master *hashing* or detail *chaining*, becomes (c) a "search field."

We refer to those classes "above" a given class as **superclasses**. The superclasses "immediately above" a given class have *a few less* properties in common with the given class. The superclasses "way above" a given class have *many less* properties in common with the given class. Every step "up" in the class hierarchy gives us less differentiation—more sameness. For example, we may consider the superclass of **bits (Binary digits)** as the superclass for all digital data structures known to humanity. It boggles my mind to think of all the

1. HP's OpenODB uses *type* instead of *class and type hierarchy* instead of *class hierarchy*.

things that we have been able to model by means of highly-specialized bunches of bits.

A given class can have zero, one, or more *subclasses*. A given class can be a subclass of zero, one, or more *superclasses*. If a class is a subclass of several superclasses, it has **multiple inheritance** properties, because it is a member of multiple class hierarchies. In this case, we must watch out for potential conflicts caused by common names—but different meanings—attached to methods, data variables, or state variables.

Reusable software

EVEN THOUGH it is a worthy goal, the idea of *reusable software* remains more of a dream than a reality.

To be able to reuse *anything*, we must be able to classify it and to retrieve it whenever we need it. If you have a friend who is a pack rat—or if you are a pack rat yourself—you know that *having something* is totally different from *being able to use that something when the need arises.*[1]

Unless we develop better systems to deal with the organization (and retrieval) of *existing* class definitions and class libraries, we will continue to suffer long training periods—more than a year, usually—for object-oriented programmers. And, even after such long training, mere humans are not meant to tax their intellectual—and memory—capabilities to such inhuman extremes.[2]

1. This is, precisely, the problem that database management systems attempt to solve.

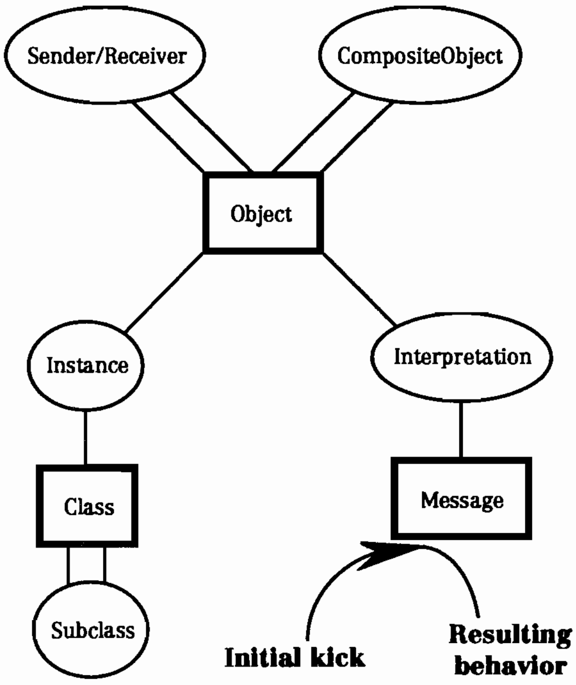
2. HP's OpenODB promises to resolve this problem by means of a graphical browser, "which allows you to navigate graphically through the database and locate reusable code."

The net result is that, despite the promise of software reusability, most programmers still re-invent the wheel on a daily basis.

A view of the object-oriented world

WE ARE NOW EQUIPPED to diagram our understanding or the object-oriented world.

There are as many views as people. This is *one* possible view on this topic:



The whole idea is that an **initial kick** to the system will originate the initial **message**. Some (hopefully) appropriate **object** will, through an act of (hopefully accurate) **interpretation**, bind the (hopefully) appropriate **method** to the message. The initial method may, in turn, result in a *cascade* of messages to other (hopefully more appropriate) objects throughout the various **class hierarchies** until, eventually, the end result will be some (hopefully appropriate) **resulting behavior**.[1]

A description of the view

WE SEE THAT this diagram has two parts: The **user**'s viewpoint and the **designer**'s perspective.

The user really wants to bother only with the lower right-hand part of the diagram—the initial kick and the resulting behavior.

The designer, besides being keenly aware of the user's needs and wants, must thoroughly understand the whole diagram.

The user's concern

THE user's concerns are:

- To provide, through some kind of (hopefully reasonable) syntax, the *initial-kick message.*
- To wait, hopefully a reasonably short time, while being entertained by some kind of background trivia regarding the various state transitions of the system.
- To get, eventually, the *resulting behavior* (which may be expected or unexpected, pleasant or unpleasant).

1. There is always the possibility of a total collapse somewhere along the way, causing the system to *crash* (with a more-or-less-polite system-failure message) or to *hang* (without giving us any clue at all).

THE designer's responsibilities include:

The designer's responsibility

• The **crafting** of *individual objects*, with their *methods* (hopefully efficient and easy to maintain), their *variables* (hopefully forming a minimal orthogonal set) and their vocabulary of *messages* (hopefully clear and to the point).

• The **organizing** of these objects—avoiding infinite cycles—along these dimensions:

1 COMPOSITIONAL: Build a **bill of materials** for objects (the "CompositeObject" relationship in the diagram). *Composite* objects consist of *component* objects. According to our model, a given object can be composed of zero, one, or more components (and a given object can form part of zero, one, or more composite objects). If possible, provide certain qualified objects with the intelligence to reconfigure the *CompositeObject* relationship at run time.

2 HIERARCHICAL: Abstract (or *factor out*) as many properties as possible—from as many objects as possible—at a time and consolidate them into **classes**. Then, organize the classes themselves into **subclasses** and **superclasses** (the "Subclass" relationship in the diagram). According to our model, a given class can be composed of zero, one, or more subclasses (and a given subclass can form part of zero, one, or more superclasses). If possible, provide certain qualified objects with the intelligence to reconfigure the *Subclass* relationship at run time.

3 COMMANDO-LIKE:[1] Establish quick chains of command (the "Sender/Receiver" relationship in

1. A *commando* is a member of a small military unit specially trained to make quick raids.

the diagram). Avoid the yo-yo effect, which causes unnecessary trips up and down the chain of command (thrashing). Try to reach the primitive objects, *where the rubber meets the road,* in the least conceivable number of steps. If possible, provide certain qualified objects with the intelligence to reconfigure the *Sender/Receiver* relationship at run time.

• The **defining of instances** that can be *overridden.*

• The **defining of interpretations** that can be *polymorphic* or *overloaded.*

Review: What's in a name?

THE name IMAGE gives us an opportunity to review some of the terms that are typical of object-oriented shoptalk.

In the beginning, we had "IMAGE/3000." This was a term that conveyed two pieces of information: (1) that the DBMS was IMAGE and (2) that the underlying computer was the HP3000, running under the MPE (*Multi Programming Executive*) operating system.

Then, there was "TurboIMAGE" for CLASSIC (or CISC, *Complex Instruction Set*) computers. Even though "turbo" connotes speed and power, a few of the changes also had to do with relaxing some of the "cargo" limitations of IMAGE/3000 (such as the number of datasets per database, the number of fields per dataset, the number of detail entries per chain, and so on). In the 1980s, we had "TurboIMAGE/XL" for RISC (or HPPA, *HP Precision Architecture*) computers. Now, with MPE'S POSIX ("Open Systems") compliance in the early 1990s, we have "TurboIMAGE/iX."

Do these different names refer to the *same* DBMS? Yes and no. Do they refer to *different* database management systems? Yes and no. From the viewpoint of a standard non-privileged user, all of these versions of IMAGE are equivalent (as long as we do not exceed the limits imposed by IMAGE/3000: 99 datasets versus 199 for TurboIMAGE, and so on). From the perspective of Adager, there are important *internal* differences among the various versions of IMAGE (the layout of the root file and the size of master chainheads, for instance). But Adager *encapsulates* these differences by means of low-level *objects* that *hide* the "minor technical details" from the high-level Adager objects.

So, are the various versions of IMAGE the same or different? "Yes, they are the same," because they belong to the same *class*.

But "no, they are different," because each is a distinct *class instance* or *object* in its own right. They are of the same *object type* but they are different *objects*.

The various versions of IMAGE share a lot in common (their *inheritance*). But each implementation allows its objects unique combinations of *state* (described by private variables) and *acceptable behavior* (characterized by *methods*).

Let's look at one example. Suppose that an IMAGE dataset has 127 fields—this fact is the dataset's *state*. Let's say that you pass a *message* to Adager: **Add Field**. Under IMAGE/3000, *Add Field* is not a legal Adager behavior from this state (IMAGE/3000 specifies MaxFields=127), whereas for TurboIMAGE *Add Field* is a legal behavior from this state (TurboIMAGE defines MaxFields=255). Adager's field-manipulating objects know all of these rules and regulations. Even more importantly, besides knowing the rules, these Adager objects know, at all times, whether the rules are *applicable* under the current circumstances.

In general, for any given state, it is possible to select from a finite set of behaviors. The *definition* of an object's legal states and the *implementation* of the legal behaviors for each state are an important part of object-oriented programming.

Review: Code and data

LET'S LOOK at some examples involving *code* (the instructions that tell the computer what to do) and *data* (the things that will be affected as a result). We may want to think of code as being oriented towards *verbs* and of data as being oriented towards *nouns*.

Review: The HP3000

REENTRANT code is a significant feature of the HP3000 computer. Only one copy of a program needs to reside in main memory, regardless of the number of users (or processes) accessing it. Each process has its own private data area but all processes running a given program share the only copy of the program. The HP3000 keeps the *intelligence* separate from the *data*.

This may appear to be at odds with the object-oriented philosophy of "consolidating code and data," but it really is not, since the HP3000 *manages* code and data in a tightly-coupled way. The advantages have proven themselves for many years now:

- Less memory requirements, for true multi-user applications, than conventional operating systems.
- Easy support for recursive procedures.
- Easy support for reentrant procedures.

Review: Binding choices on the HP3000

THE HP3000 COMPUTER, through its MPE operating system, offers *several* choices for binding.

There is the basic stuff, which other operating systems such as UNIX & MS-DOS also have: USLs (user segmented libraries) which are repositories for compiler output, to be bound at link time.

At run time, MPE has SLs (segmented libraries), arranged in a hierarchy that allows overriding (the user's logon group overrides the user's logon account, which in turn overrides the system). MPE also provides a convenient procedure, *LoadProc,* that allows you to dynamically load, at run time, any procedure from any segmented library in your hierarchy. You do not have to recompile any of your programs and you can add & delete library procedures at any time, when they are not loaded.

MPE, on its XL version and on its POSIX-compliant iX version, offers XLs (analogous to DLLs, Dynamic Link Libraries[1] in MS-WINDOWS, which were not available in plain MS-DOS—where you had to link everything at once, and only once: at the beginning).[2]

Review: Self-describing files

FILE USER LABELS are an old idea. I think IBM pioneered labeled magnetic tapes, which the HP3000 has supported for a long time. Most people know this. What few people know is that the HP3000, under all versions of its MPE operating system, has always supported user labels on *disc* files.[3] David Greer, of Robelle Consulting, has written a paper, "*Adopting self-describing files,*" that discusses—in great depth—one application of user labels on the HP3000 computer.[4]

User labels *always* go along with the file, but we can access them only through special intrinsic procedures. This has a dual advantage:

1. Tom Keffer notes: In DLLs, each process does *not* have its own space. Processes have their own stacks, but they share a single data segment... ugh!

2. UNIX seems to be running a bit behind in this area. As a matter of fact, Bill Gates of Microsoft has vowed to make its WINDOWS-NT (new technology) a formidable competitor for UNIX. The next few years should be exciting!

3. Fred White and his original IMAGE team in Hewlett-Packard certainly knew about this useful feature. Every IMAGE dataset and root file contains user labels that keep track of important database information.

4. As far as I know, neither MS-DOS nor UNIX support this construct.

• We can keep "the data" in the file *and* "the information about the data" ***(the metadata)*** in the file's user label. The two sides of the coin are encapsulated by the operating system and we don't have to worry about version control, consistency, congruency, and other bits of management.

• The "normal" user does not even know about—and is not bothered by—the existence of the metadata. The "systems" user can easily access the metadata, as well as the data, at any time.

Keeping data together with its metadata is an important element of the object-oriented philosophy.

Review: PostScript

LET'S LOOK at another example from the standard in electronic typography: *PostScript.* This standard keeps its data *(fonts)* separate from its code *(the PostScript interpreter with its rendering technology).* "This means that rendering technologies can improve without obsoleting the existing Type 1 font library... The future will doubtless bring new imaging technology that imposes fresh criteria for rendering typefaces... Type technologies that rely on rendering intelligence tightly coupled to the character data are ill-suited to adapting smoothly to future requirements."[1]

Review: Apple's Macintosh

DID YOU THINK that I had forgotten the Macintosh? How could I, given the fact that I have typed—and typeset—this paper on a Macintosh, using all kinds of object-oriented tools? The Macintosh is, after all,

1. Grosvenor et al (1992). *The PostScript Font Handbook*. Addison-Wesley.

the first commercially available computer that was conceived and built using the object-oriented paradigm. But we must end our initial exploration at some point and this, I think, is a good point to end this paper.

Farewell

HOPEFULLY, you will continue your own exploration of the object-oriented world. Have a pleasant journey—and please share some of your findings with me!

Gratitude

I AM FORTUNATE to have friends who share my enthusiasm. I would like to express my gratitude to all of them—particularly to the following.

Berni Reiter, of Objective, Inc., for being Mrs. Objective herself. Wirt Atmar, of AICS Research, for his vision, deep understanding, and encyclopædic knowledge. René Woc, of Adager, for his keen eye. David Greer, of Robelle Consulting, for our conversations on self-describing files, on Turbo Pascal, and on our object-oriented implementations within the rules of the SPL and SPLASH programming languages. Stan Sieler, of Allegro Consultants, for his SPL and SPLASH "$INCLUDE" packages—and for his contributions to the SPLASH compiler. Tom Keffer, of Rogue Wave Software, for squeezing technical discussions during a family visit to Sun Valley. Ross Scroggs, of Telamon, for his cheerfulness while sharing thoughts on MPE, SPL, UNIX, C, and telecommunications under *every* known protocol—and even under some unknown ones. Fred White, of Adager—formerly of Hewlett-Packard, where he was a member of the original IMAGE team—, for his keen eye, which is in the same league as René's.

Generic Routines In COBOL Using $DEFINE And .LEN.

By
Rick Hoover
CIV Software
700 Hanover Dr
Shelby, NC 28150

HP has given COBOL programmers two powerful but seldom used statements, the $DEFINE and .LEN. statements. The $DEFINE statement allows a wonderful way to build generic routines. The .LEN. pseudo-intrinsic will calculate record lengths. This paper will document actual programming concepts that will allow programmers a wealth of routines in a very short time by using these two "underused" conventions.

.LEN. pseudo-intrinsic

The .LEN. pseudo-intrinsic calculates the length of tables, items and groups. By using the .LEN. call, programs can be left alone from re-calculating record lengths. Record lengths are passed back from the call. Any program or intrinsic (such as VPLUS or IMAGE) that needs to know the size of a buffer can use this type of routine. If a copy library contains the record layout without the size, the program will need to contain the record size. If, at a later date the record layout changes (especially if the layout gets bigger) all programs will need to be checked for changes. And, making sure that the calculations are correct for sizing must be perfect. Use the .LEN. statement to force a correct record size every time. To use the .LEN. statement:

```
01 MY-RECORD.
        05 MY-FIELD1            PIC X(10)
        05 MY-FIELD2            PIC S9(04) COMP.
        05 MY-FIELD3.
                10 MY-FIELD4    PIC X(10).
                10 MY-FIELD5    PIC S9(04) COMP.
...
01 LEN-RECORD-LENGTH            PIC S9(04) COMP.

CALL INTRINSIC ".LEN." USING MY-RECORD GIVING LEN-RECORD-LENGTH
CALL "VPUTBUFFER" USING COMAREA V-RECORD LEN-RECORD-LENGTH
```

As you can see, the record layout format doesn't matter. No matter how the layout is defined, the .LEN. intrinsic will calculate the size for the program. This allows the programmer to not hard code the record size in any program which means that no variable will need to be changed.

$DEFINE Statement

The $DEFINE statement is placed in the program to give the program a generic routine. In effect, what you create is a macro for COBOL with the ability to have replaceable parameters. The format of a $DEFINE is:

```
$DEFINE macroname = stringtext#
```

The $DEFINE starts in column 7. The macroname must first have a non-alphanumeric character (default is the percent sign %) followed by standard COBOL naming conventions. An equal sign begins the macro definition. The macro definition is standard COBOL code in standard formats. For example:

```
$DEFINE %ADDFIELDS = ADD 1 TO LINE-COUNTER
                     ADD LINE-COUNTER TO TOTAL-COUNTER
                     DIVIDE TOTAL-COUNTER BY 4 GIVING AVERAGE-COUNTER.#
```

The PROCEDURE DIVISION will contain:

```
        %ADDFIELDS
```

Which will replace the %ADDFIELDS in the PROCEDURE DIVISION with the COBOL code defined in the $DEFINE statement. Because the COBOL code immediately follows the equal sign, the code will exist on the same line as the macro statement. If, for example, the macro definition looked like this:

```
$DEFINE %ADDFIELDS =
                     ADD 1 TO LINE-COUNTER
                     ADD LINE-COUNTER TO TOTAL-COUNTER
                     DIVIDE TOTAL-COUNTER BY 4 GIVING AVERAGE-COUNTER.#
```

then the COBOL code will end up on the line following the macro call.

Now this sounds ok to me but the power of the macro calls are not being used to their fullest. The $DEFINE allows up to 9 replaceable parameters in the macro definition. To use the $DEFINE with a macro replacement use the following format:

```
$DEFINE %ADDFIELDS =
        ADD 1 TO !1
        ADD !1 TO !2
        DIVIDE !2 BY 4 GIVING !3#
```

Then The PROCEDURE DIVISION statement will be:

```
%ADDFIELDS(LINE-COUNTER#,TOTAL-COUNTER#,AVERAGE-COUNTER#)
```

The generated COBOL code will become:

```
ADD 1 TO LINE-COUNTER
ADD LINE-COUNTER TO TOTAL-COUNTER
DIVIDE TOTAL-COUNTER BY 4 GIVING AVERAGE-COUNTER
```

BY using the replaceable parameters, you can create many different types of generic routines. For example, let's say that you have a program that needs to access 3 different IMAGE manual master data sets. You can either write the DBGET code 3 times with the different parameters or use a $DEFINE statement to control the DBGET.

```
$DEFINE %DEFINEDBGET =
        CALL "DBGET" USING !1
                           !2
                           IMAGE-MODE-7
                           IMAGE-STATUS
                           IMAGE-ALL-LIST
                           !4
                           !3#
```

Then in the procedure division put in the following code:

```
%DEFINEDBGET(SAMPLE#,
             ORDER-MASTER#,
             ORDER-NUMBER#,
             ORDER-RECORD#)
```

Where the data base name is SAMPLE, the data set name is ORDER-MASTER, the ORDER-NUMBER is the argument and the ORDER-RECORD is the WORKING-STORAGE area that will contain the record contents after the successful DBGET.

Did you notice that the fields are not in the same order as the DBGET requires? Remember, you are creating the macro, not the actual call. You may place the replaceable parameters in any order you wish. You can also use the parameters over again in the macro (as in the prior example).

As you can probably realize by now, the replaceable parameters must end with a pound sign (#) to delineate them from each other. This is necessary because you can pass text in the parameters as in:

```
%COMPAREVAL(A = B#)
```

This will pass to the macro the logical statement A = B. Without the pound sign to delineate the parameters, the macro would assume that there was 3 parameters to pass.

You may also pass a null value to the macro by passing a pound sign with no value, as in:

```
%MYVAR(A = B#,#,300)
```

Which will pass A = B as the first parameter, 300 as the third parameter and null as the second parameter.

<u>BIG RULE</u>

There can be no spaces between the macroname and the parenthesis. In other words:

```
%MYDEFINE(APPLE#,BANANA#,ORANGE#)
```

is valid, but

```
%MYDEFINE  (APPLE#,BANANA#,ORANGE#)
```

is not because there is a space between the MYDEFINE and the parenthesis.

You can also have more parameters than needed as in the following example:

```
%DEFINE SHOWSTMT = DISPLAY !1,!2,!3,!5.#
```

```
%SHOWSTMT("A"#,"B"#,"C"#,"D")
```

This will generate:

```
DISPLAY "A","B","C",.
```

Since the "A" replaces the !1 parameter, "B" replaces the !2 parameter, "C" replaces the !3 parameter. "D" does not show up because it is the !4 parameter replacement and since !5 does not exist in the PROCEDURE DIVISION.

Finally, what if you need to use either the !, #, or % in a macro? There is one other command that is available to use to set up any of the above as non-default value.

```
$PREPROCESSOR KEYCHAR=%,PARMCHAR=!,DELIMITER=#
```

This will allow you to use other characters in the macros. The example above shows the default values. To change them:

```
$PREPROCESSOR KEYCHAR=~,PARMCHAR=@,DELIMITER=&
$DEFINE ~MOVEIT=MOVE @1 TO @2&
```

You can see how the changes can be made. Be aware that once you use the $PREPROCESSOR command, you should take care that the new values will work and that other $DEFINES will use the same changes.

This short paper should shed some light on 2 underused commands in HP COBOL. You can set up many of the routines that are drudgery now. You can also set up these routines in copy libraries for use or use the $INCLUDE command to insert the code.

Database freedom

F. Alfredo Rego

Adager Corporation
Sun Valley, Idaho 83353-0030, USA

Telephone +1 (208) 726-9100, Fax +1 (208) 726-8191

YOU WANT TWO KINDS OF FREEDOM when dealing with a database:

• You want to be free from having to wait forever for the result of a query.

• You want to be free from having to change all your programs when you change the layout of your database.

These freedoms are not free. You have to earn them. I discuss several database traps and I describe strategies that you can use to avoid being trapped.

When I presented my abstracts for the 1992 INTEREX Conference in New Orleans, I had planned to write two different papers:

1 Database freedom

2 Object-oriented methodology: the Adager instance

When I actually wrote the papers, I decided to consolidate them. So, during the *Database freedom* presentation, I will discuss the appropriate subset of ideas from my *Object-oriented methodology* essay, which you can find in these proceedings as paper number 3117.

Paper 3152

Migration from TurboIMAGE to ALLBASE

by

George B. Scott, President
Great Business Solutions, Inc
P.O. Box 950
Graham, WA 98338-0950
(206) 847-8924

Introduction

This paper describes the steps and activities involved in migrating an application from TurboIMAGE to ALLBASE, Hewlett-Packard's, (HP's), relational data base product. Many of these same steps are relevant if one is not migrating an application, but instead is developing a new application using ALLBASE as the data base tool.

This paper will present some issues related to project administration to highlight special data base migration issues. The bulk of the paper is oriented around data base design. Other areas discussed are data loading, data clean-up and documentation. So without further ado, let us start with the first step, which is to decide how the project will be managed / administered.

Project Administration

It is not altogether uncommon that a project has a goal or target which is stated in very general terms such as "Improve our service to users by building an integrated help desk system." Such goals have been known to be more pointedly defined by adding requirements such as "Relational technology (SQL) must be used for all future data base development."

This example has been chosen because it represents a fairly simple case which can transcend the many application backgrounds present in the HP user community. Hopefully, the techniques discussed herein and the problem areas highlighted will be relevant to whatever application area is of interest to you.

An early problem presented to project administration is to translate the broad and general project objective into a plan which has sub-tasks, required resources, and a schedule. Obviously, all of ***our*** problems are much greater than the above mentioned goal, so you will need some project management tool such as Microsoft's PROJECT or

Symantec's TIMELINE or any of the other myriad of fine project management tools available today (the mention of any product or tool in this paper is not a recommendation, but is done for illustrative purposes only).

Another key utility is the word processor which will be used to record narrative about the tasks, to write technical reports, to write memo's, etc. It is quite helpful if the tool selected is commonly used by all members of the migration team so that you can share files.

A third key utility is some type of CASE tool which provides the ability to draw a schematic data base layout, especially to show referential integrity relationships. Other capabilities are the documentation of tables and columns with narrative descriptions. It is quite helpful if this tool also allows you to correlate problems and requirements to the tables, columns, referential integrity linkages, indexes, views and other ALLBASE constructs.

The fourth utility frequently required is a data dictionary which is used by a screen handler / report writer. If this is not the same as provided by the CASE tool selected, then some type of interface utility will probably be needed to synchronize the two in order to insure correspondence of structure, definitions and attributes.

Obviously, the more tightly the above four tools are linked and the closer the look and feel among the tools, the easier your project will be to manage and the less manual effort and time will have to be spent to insure all pieces of the design and documentation are in agreement with each other.

The last principal piece of the puzzle is the ALLBASE schema itself, which contains all the structure, relationships, access and security rules embedded within it.

On the other hand, it has been my experience that a schematic coupled with a list of tables layouts in alphabetical order and column definitions in alphabetical order make up a much more useful tool set than a schema which is difficult even for propeller heads like me to use.

Thus, use a picture where ever possible to communicate data base structure / design. Use a simplified table layout when you are talking about the contents of a table. Avoid confusing users with referential integrity linkages, indexes, views and other SQL buzzwords. In short, don't try and confuse them with the facts. Instead, try to help them understand the functionality and improved service which will be provided by the new "integrated help desk".

Elementary Design

After twenty years of data base design experience (and mistakes) and having read many books and papers, I have developed some practices which I hope will be helpful to you. The first suggestion is to understand the subject area in which you are to work. Whenever I have followed some existing design without the accompanying application insight, I seem to have to redo some piece of the puzzle. The reason for caution at this point is the impact on the project schedule is much less if extra time is spent at the concept stage as opposed to re-writing code and the associated documentation. Besides, the people driving the project are usually more lenient in the beginning. If you find yourself in a position where you must follow someone else's design at the beginning, just grin and bear it.

After having developed some perspective on the real world problem which you are trying to solve, I suggest that you then sit down with a large piece of paper or white board and just write down the primary entities which you think you need to track. For example, in this scenario you might come up with the following list:

USERS	**EQUIPMENT**	**SOFTWARE**
SUPPORTERS	**PROBLEMS**	**RESPONSES**

The important thing at this point is to try and avoid thinking about attributes. After additional thought, you might want to add **TRAINING** to the above list. For simplification in this exercise, I have included networking software in the software category and networking hardware in the hardware category. If we sketch these out on a piece of paper they may look something like Figure 1. This is a good start.

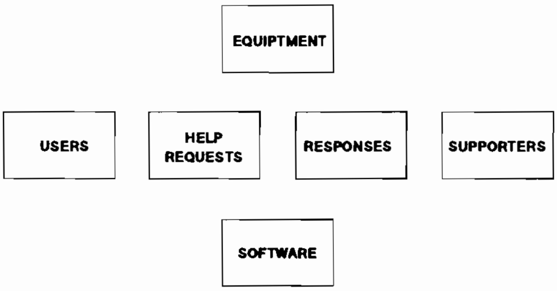


Figure 1 -- Major Primary Entities

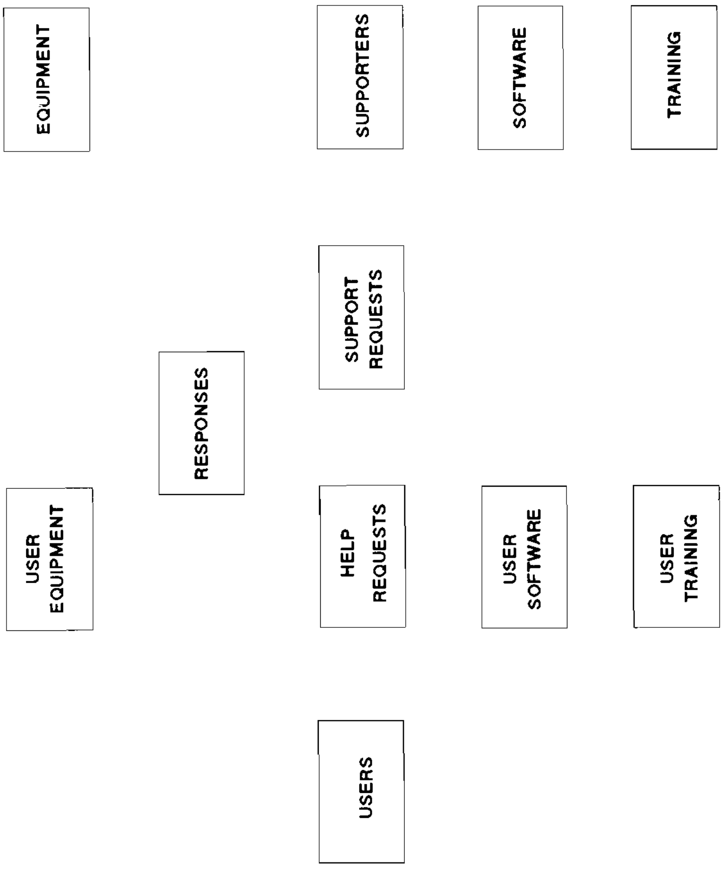


Figure 2 -- Primary Entities and Cross-References

Expanding the Design

The next phase is to consider the situations where more than one-to-one relationships exist. For example, many pieces of software or hardware may be associated with a user; multiple problems (hereafter referred to as help requests) may be associated with a particular user, piece of equipment or piece of software; multiple support resources may be requested for help on a particular help request and; multiple responses may be made for each help request.

One aspect related to our "help desk" design that hasn't been discussed yet is that the help desk is usually attended by a very few people who are often able to answer the simple questions, but who utilize additional support resources when they cannot answer more difficult questions. Some organizations want all requests initially funneled through the help desk personnel, who may direct the users to contact other support personnel. Other organizations want all external contacts to be made by the help desk personnel. Some support vendors, such as HP, have coerced customers to have a very small list of authorized callers for support. Other companies, such as Microsoft, don't care who you are as long as you have their product legitimately.

The bottom line is that requests for help are sure to be funneled through some support team which is probably understaffed; thus, it is important to make the software facilitate getting answers more quickly, and to track and alert the personnel to situations where requests for help remain in an "open" status for any extended period of time. It has been my experience that most people are quite reasonable and patient if they believe you are working ***their*** problem. Using the above information, the next step is to add the following tables to our design:

USER_EQUIP	=	a user, equipment cross reference
USER_SOFTWARE	=	a user, software cross reference
USER_TRAINING	=	a user, training cross reference
SUPPORT_REQUESTS	=	a help request / support request cross reference

The final piece of the puzzle that is usually helpful to have is some concept of a fixed asset type of information. Although this could have some very limited value to accounting, it will probably have a great deal of value to whoever is responsible for implementing upgrades or gross replacements. I remember a case a few years ago when approximately 70 PC's were purchased with a particular hard drive. Within a few weeks, these particular drives started dying violently. It rapidly became apparent that it would be cheaper to replace them all than to see a large number of electrical engineers redo the last several days (weeks) of work because they hadn't backed up their systems. Lets face it, if backups aren't automatic or aren't done by someone else, they probably won't get done as often as they should. This likelihood diminishes as the level of management rises. The question was "Who has a list of the current location for each of the defective PCs?" Unfortunately, the answer was "No one."

Adding Columns to Data Tables

At this point you have probably starting thinking about what attributes are associated with what entities and have probably starting writing down on your paper the table names, the primary key which defines the uniqueness of the table and the attributes related to the primary key. Normalization is usually done at this stage. The following shows columns which might be selected:

USERS	**EQUIPMENT**	**SOFTWARE**
USER_ID	EQUIP_ID	SOFT_ID
USER_LASTNAME	EQUIP_TYPE	SOFT_NAME
USER_FIRSTNAME	VENDOR_MFG	VENDOR_MFG
USER_PHONE	VENDOR_PUR	VENDOR_PUR
USER_LOCATION	EQUIP_MODEL	SOFT_RELEASE
	SERIAL_NUM	SERIAL_NUM
	DATE_PUR	DATE_PUR
	PO_NUM	PO_NUM
	EQUIP_STATUS	SOFT_STATUS
	DATE_STATUS	DATE_STATUS
	UNIT_COST	UNIT_COST
	UNIT_SIZE	SOFT_CLASS

CONFIGURATIONS	**USER_SOFTWARE**	**USER_EQUIPMENT**
CONFIG_ID	USER_ID	USER_ID
CONFIG_DESC	SOFT_ID	EQUIP_ID
CONFIG_STATUS	CONFIG_ID	CONFIG_ID
DATE_STATUS	OWN_SHARE	OWN_SHARE

USER_TRAINING	**TRAINING**	**TRAINERS**
USER_ID	TRAIN_ID	TRAINER_ID
TRAIN_ID	CLASS_NAME	TRAINER_NAME
UNIT_COST	DATE_CLASS	TRAINER_PHONE
	TRAINER_ID	TRAINER_CORP
	CLASS_COST	

HELP_REQUESTS	**SUPPORT_REQUESTS**	**RESPONSES**
HELP_REQ_ID	SUPP_REQ_ID	RESP_ID
USER_ID	HELP_REQ_ID	HELP_REQ_ID
DATE_TIME_REQ	DATE_TIME_REQ	SUPP_REQ_ID
DATE_TIME_RESP1	DATE_TIME_RESP1	NARR_ID
HELP_STATUS	SUPP_STATUS	
DATE_TIME_STATUS	DATE_TIME_STATUS	
NARR_ID	NARR_ID	
	SUPR_ID	

EQUIPMENT_ SUPPORTERS
EQUIP_ID
SUPR_ID
CONTRACT_ID
DATE_CONTRACT
DATE_TERM
CONTRACT_COST
TIME_AVAIL
VEND_ID

SOFTWARE_ SUPPORTERS
SOFT_ID
SUPR_ID
CONTRACT_ID
DATE_CONTRACT
DATE_TERM
CONTRACT_COST
TIME_AVAIL
VEND_ID

SUPPORTERS
SUPR_ID
SUPR_NAME
SUPR_PHONE
SUPR_CORP

VENDORS
VEND_ID
VEND_NAME
VEND_STREET
VEND_CITY
VEND_ST
VEND_ZIP
VEND_PHONE
VEND_CONTACT

CONTROL_DATA
SYS_ID
NEXT_CONFIG_ID
NEXT_EQUIP_ID
NEXT_HELP_ID
NEXT_NARR_ID
NEXT_SOFT_ID
NEXT_SUPP_ID
NEXT_SUPR_ID
NEXT_USER_ID
NEXT_VEND_ID

NARRATIVE
NARR_ID
DATE_TIME_NARR
NARR_CMT

The next step is a sort of oscillation between checking the drawing (See Figure 3) and the list of items to determine if the table layouts and the schematic are in agreement with one another. I would suggest that you audit each against the other until you can find no further discrepancies. I usually like to start with the drawing and check the tables to insure that each table has the key items required for the referential linkage. That done, I usually check each table against the drawing to insure the drawing has the appropriate referential lines and that they are pointed in the correct direction.

Referential Integrity Linkages

A referential integrity linkage exists in ALLBASE when one table is linked to another such that one, and only one, entry exists in the referenced table for each value in the referencing table. In other words, the value references must be ***unique*** in the referenced table. It is very similar to the concept of a master set in TurboIMAGE, with the major difference being that an ALLBASE table can have more than one unique identifier and more than one index. Within the above table layouts, the EQUIP_ID would be unique within EQUIPMENT. Also unique would be the combination of fields (EQUIP_TYPE, EQUIP_MFG, EQUIP_MODEL, SERIAL_NUM). Rather than replicate the four fields throughout the data base design, the EQUIP_ID field is used. Either way works.

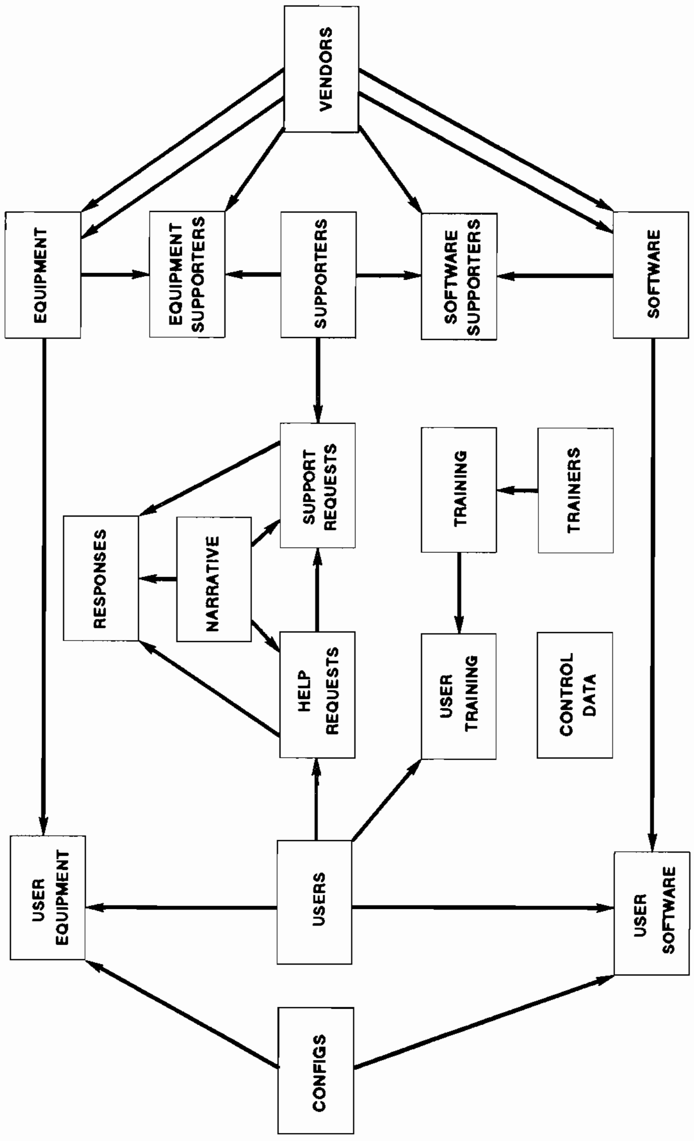


Figure 3-- Entities and Relationships

Primary Key

Note that in ALLBASE, the primary key may be composed of ***multiple*** columns. It is not necessary to have a concatenated key defined for each access path you want to use. You can define an index on any combination of columns. Primary keys can also have a name associated with them, or else ALLBASE will assign its own name when the key is added. If you are using such a product as QUICK by COGNOS, I highly recommend that you assign the name, since a change in the data base structure can cause the ALLBASE assigned name to change.

Building the ALLBASE Schema

The ALLBASE schema is quite similar to the concept of a TurboIMAGE schema, except that the ITEMS part is missing. Within the ALLBASE schema is information relating what tables are to use what MPE files (many tables can use a single file, but each page of a file can have only information related to one table in it).

This is the point when you start looking at the various TurboIMAGE schemas and observing that not only do the various items in the schema not quite fit the definition of terms you had in mind, but that different sizes were used between the different TurboIMAGE sources. I highly recommend that at this point you ***check the actual data in the TurboIMAGE data bases!*** It is rather amazing that we can sometimes go to great effort to figure out how to handle a particular conversion, only to learn that the source data field has never been used, or worse, has been used to store data that is different from the field definition.

At this point, I suggest you build your ALLBASE schema. The schema in Figure 4 is one which was built to demonstrate some of the various constructs which you will need to know: primary keys, referential checks and indexes. It also shows several of the various data types available to you.

One of the joys of creating an ALLBASE SQL data base is debugging the schema. The first joy you get to experience is that the utility used to create the data base is a parser that usually gives up as soon as it finds a single error in a table. Thus, for blind people such as myself who on occasion have trouble distinguishing between a comma and a period, it usually takes several passes to get the syntax exactly correct. This schema took about 8 passes (10 min.). It's not the time, it's the frustration level that makes it such a memorable joy. Of course, you get to use a different utility to purge the data base (SQLUTIL "PURGEALL"), rather than just being able to enter ISQL and purge the old garbage, and then attempt to create a new "perfect" data base. Perhaps this is what is meant by "double your joy, double your fun".

```
SET ECHO ON;
/***************************************************************/
/*    SQL SCHEMA: HLDESK                                       */
/*          TYPE: PRODUCTION                                   */
/*   RELEASED BY: George B. Scott                              */
/*                                                             */
/* Release History:                                            */
/* VERSION   DATE REL   REL# WHO REASON                        */
/* ======== =========== ==== === ========================== */
/* 00.00.00 24-MAY-1992 P001 GBS New System                    */
/***************************************************************/

START DBE 'HLDESK' MULTI NEW
      BUFFER = (15, 24) ,
      TRANSACTION = 5,
      DBEFILE0 DBEFILE US
         WITH PAGES = 1000,
         NAME = 'HL000',
      LOG DBEFILE DBELOG1
          WITH PAGES = 2000,
          NAME = 'HL001L';


CREATE DBEFILESET FS_HELP;
CREATE DBEFILE HL001T
      WITH PAGES = 2000,
      NAME = 'HL001T',
      TYPE = TABLE;

CREATE DBEFILE HL001X
      WITH PAGES = 4000,
      NAME = 'HL001X',
      TYPE = INDEX;


ADD DBEFILE HL001T TO DBEFILESET FS_HELP;
ADD DBEFILE HL001X TO DBEFILESET FS_HELP;
COMMIT WORK;

CREATE PUBLICREAD TABLE HL.CONTROL_DATA
  (SYS_ID               CHAR( 4) NOT NULL PRIMARY KEY
       CONSTRAINT PK_CONTROLDATA,
   NEXT_CONFIG_ID       CHAR( 8) NOT NULL,
   NEXT_EQUIP_ID        CHAR( 8) NOT NULL,
   NEXT_HELP_ID         CHAR( 8) NOT NULL,
   NEXT_NARR_ID         CHAR( 8) NOT NULL,
   NEXT_SOFT_ID         CHAR( 8) NOT NULL,
   NEXT_SUPP_ID         CHAR( 8) NOT NULL,
   NEXT_SUPR_ID         CHAR( 8) NOT NULL,
   NEXT_USER_ID         CHAR( 8) NOT NULL,
   NEXT_VEND_ID         CHAR( 8) NOT NULL)
     IN FS_HELP;

CREATE PUBLICREAD TABLE HL.CONFIGS
  (CONFIG_ID            CHAR( 8) NOT NULL PRIMARY KEY
       CONSTRAINT PK_CONFIGS,
   CONFIG_DESC          CHAR(32) NOT NULL,
   CONFIG_STATUS        CHAR( 8) NOT NULL,
   DATE_STATUS          DATE)
      IN FS_HELP;
```

Figure 4 -- HLDESK Schema (Page 1 of 5)

```
CREATE PUBLICREAD TABLE HL.NARRATIVE
  (NARR_ID               CHAR( 8) NOT NULL PRIMARY KEY
      CONSTRAINT PK_NARRATIVE,
   DATE_TIME_NARR        DATETIME,
   NARR_CMT              VARCHAR(3600))
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.SUPPORTERS
  (SUPR_ID               CHAR( 8) NOT NULL PRIMARY KEY
      CONSTRAINT PK_SUPPORTERS,
   SUPR_NAME             CHAR(32) NOT NULL,
   SUPR_PHONE            CHAR(12) NOT NULL,
   SUPR_CORP             CHAR( 8) NOT NULL)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.TRAINERS
  (TRAINER_ID            CHAR( 8) NOT NULL PRIMARY KEY
      CONSTRAINT PK_TRAINERS,
   TRAINER_NAME          CHAR(32) NOT NULL,
   TRAINER_PHONE         CHAR(12) NOT NULL,
   TRAINER_CORP          CHAR( 8) NOT NULL)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.USERS
  (USER_ID               CHAR( 8) NOT NULL PRIMARY KEY
      CONSTRAINT PK_USERS,
   USER_LASTNAME         CHAR(16) NOT NULL,
   USER_FIRSTNAME        CHAR(16) NOT NULL,
   USER_PHONE            CHAR(12) NOT NULL,
   USER_LOCATION         CHAR(16) NOT NULL)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.VENDORS
  (VEND_ID               CHAR( 8) NOT NULL PRIMARY KEY
      CONSTRAINT PK_VENDORS,
   VEND_NAME             CHAR(32) NOT NULL,
   VEND_STREET           CHAR(32) NOT NULL,
   VEND_CITY             CHAR(16) NOT NULL,
   VEND_ST               CHAR( 2) NOT NULL,
   VEND_ZIP              CHAR(10) NOT NULL,
   VEND_PHONE            CHAR(12) NOT NULL,
   VEND_CONTACT          CHAR(32) NOT NULL)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.EQUIPMENT
  (EQUIP_ID              CHAR( 8) NOT NULL PRIMARY KEY
      CONSTRAINT PK_EQUIPMENT,
   EQUIP_TYPE            CHAR(32) NOT NULL,
   EQUIP_MODEL           CHAR(16) NOT NULL,
   UNIT_SIZE             CHAR( 8) NOT NULL,
   SERIAL_NUM            CHAR(16) NOT NULL,
   VENDOR_MFG            CHAR( 8) NOT NULL
       REFERENCES HL.VENDORS(VEND_ID),
   VENDOR_PUR            CHAR( 8) NOT NULL
       REFERENCES HL.VENDORS(VEND_ID),
   DATE_PUR              DATE,
   PO_NUM                CHAR( 8) NOT NULL,
   UNIT_COST             DECIMAL( 7,0) NOT NULL,
   EQUIP_STATUS          CHAR( 8) NOT NULL,
   DATE_STATUS           DATE)
      IN FS_HELP;
```

Figure 4 -- HLDESK Schema (Page 2 of 5)

```
/****************************************************************/
/* End of Primary Tables, Start of Secondary Tables            */
/****************************************************************/

CREATE PUBLICREAD TABLE HL.SOFTWARE
  (SOFT_ID                CHAR( 8) NOT NULL PRIMARY KEY
       CONSTRAINT PK_SOFTWARE,
   SOFT_CLASS             CHAR( 8) NOT NULL,
   SOFT_NAME              CHAR(32) NOT NULL,
   SOFT_RELEASE           CHAR(16) NOT NULL,
   SERIAL_NUM             CHAR(16) NOT NULL,
   VENDOR_MFG             CHAR( 8) NOT NULL
        REFERENCES HL.VENDORS(VEND_ID),
   VENDOR_PUR             CHAR( 8) NOT NULL
        REFERENCES HL.VENDORS(VEND_ID),
   DATE_PUR               DATE,
   PO_NUM                 CHAR( 8) NOT NULL,
   UNIT_COST              DECIMAL( 7,0) NOT NULL,
   SOFT_STATUS            CHAR( 8) NOT NULL,
   DATE_STATUS            DATE)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.TRAINING
  (TRAIN_ID               CHAR( 8) NOT NULL PRIMARY KEY
       CONSTRAINT PK_TRAINING,
   CLASS_NAME             CHAR(32) NOT NULL,
   DATE_CLASS             DATE     NOT NULL,
   TRAINER_ID             CHAR( 8) NOT NULL
        REFERENCES HL.TRAINERS(TRAINER_ID),
   CLASS_COST             DECIMAL(7,0) NOT NULL)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.HELP_REQUESTS
  (HELP_REQ_ID            CHAR( 8) NOT NULL PRIMARY KEY
       CONSTRAINT PK_HELPREQUESTS,
   USER_ID                CHAR( 8) NOT NULL
       REFERENCES HL.USERS(USER_ID),
   DATE_TIME_REQ          DATETIME NOT NULL,
   DATE_TIME_RESP1        DATETIME NOT NULL,
   HELP_STATUS            CHAR( 8) NOT NULL,
   DATE_TIME_STATUS       DATETIME NOT NULL,
   NARR_ID                CHAR( 8) NOT NULL
       REFERENCES HL.NARRATIVE(NARR_ID))
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.SUPPORT_REQUESTS
  (SUPP_REQ_ID            CHAR( 8) NOT NULL PRIMARY KEY
       CONSTRAINT PK_SUPPORTREQUESTS,
   HELP_REQ_ID            CHAR( 8) NOT NULL
       REFERENCES HL.HELP_REQUESTS(HELP_REQ_ID),
   DATE_TIME_REQ          DATETIME NOT NULL,
   DATE_TIME_RESP1        DATETIME NOT NULL,
   SUPP_STATUS            CHAR( 8) NOT NULL,
   DATE_TIME_STATUS       DATETIME NOT NULL,
   NARR_ID                CHAR( 8) NOT NULL
       REFERENCES HL.NARRATIVE(NARR_ID),
   SUPR_ID                CHAR( 8) NOT NULL
       REFERENCES HL.SUPPORTERS(SUPR_ID))
      IN FS_HELP;
```

Figure 4 -- HLDESK Schema (Page 3 of 5)

```
/*************************************************************/
/* End of Secondary Tables, Start of Tables Pointed To Only */
/*************************************************************/

CREATE PUBLICREAD TABLE HL.EQUIPMENT_SUPPORTERS
  (EQUIP_ID               CHAR( 8) NOT NULL
       REFERENCES HL.EQUIPMENT(EQUIP_ID),
   SUPR_ID                CHAR( 8) NOT NULL
       REFERENCES HL.SUPPORTERS(SUPR_ID),
   VEND_ID                CHAR( 8) NOT NULL
       REFERENCES HL.VENDORS(VEND_ID),
   CONTRACT_ID            CHAR(16) NOT NULL,
   DATE_CONTRACT          DATE     NOT NULL,
   DATE_TERM              DATE     NOT NULL,
   TIME_AVAIL             CHAR( 8) NOT NULL,
   CONTRACT_COST          DECIMAL(7,0) NOT NULL,
      PRIMARY KEY (EQUIP_ID, SUPR_ID)
       CONSTRAINT PK_EQUIPSUPPORTERS)
       IN FS_HELP;

CREATE PUBLICREAD TABLE HL.SOFTWARE_SUPPORTERS
  (SOFT_ID                CHAR( 8) NOT NULL
       REFERENCES HL.SOFTWARE(SOFT_ID),
   SUPR_ID                CHAR( 8) NOT NULL
       REFERENCES HL.SUPPORTERS(SUPR_ID),
   VEND_ID                CHAR( 8) NOT NULL
       REFERENCES HL.VENDORS(VEND_ID),
   CONTRACT_ID            CHAR(16) NOT NULL,
   DATE_CONTRACT          DATE     NOT NULL,
   DATE_TERM              DATE     NOT NULL,
   TIME_AVAIL             CHAR( 8) NOT NULL,
   CONTRACT_COST          DECIMAL(7,0) NOT NULL,
      PRIMARY KEY (SOFT_ID,  SUPR_ID)
       CONSTRAINT PK_SOFTSUPPORTERS)
       IN FS_HELP;


CREATE PUBLICREAD TABLE HL.RESPONSES
  (RESP_ID                CHAR( 8) NOT NULL PRIMARY KEY
       CONSTRAINT PK_RESPONSES,
   HELP_REQ_ID            CHAR( 8) NOT NULL
       REFERENCES HL.HELP_REQUESTS(HELP_REQ_ID),
   SUPP_REQ_ID            CHAR( 8) NOT NULL
       REFERENCES HL.SUPPORT_REQUESTS(SUPP_REQ_ID),
   NARR_ID                CHAR( 8) NOT NULL
       REFERENCES HL.NARRATIVE(NARR_ID))
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.USER_EQUIPMENT
  (USER_ID                CHAR( 8) NOT NULL
       REFERENCES HL.USERS(USER_ID),
   EQUIP_ID               CHAR( 8) NOT NULL
       REFERENCES HL.EQUIPMENT(EQUIP_ID),
   CONFIG_ID              CHAR( 8) NOT NULL
       REFERENCES HL.CONFIGS(CONFIG_ID),
   OWN_SHARE              CHAR( 4) NOT NULL,
      PRIMARY KEY (USER_ID, EQUIP_ID, CONFIG_ID)
       CONSTRAINT PK_USEREQUIPMENT)
      IN FS_HELP;
```

Figure 4 -- HLDESK Schema (Page 4 of 5)

```
CREATE PUBLICREAD TABLE HL.USER_SOFTWARE
  (USER_ID                CHAR( 8) NOT NULL
       REFERENCES HL.USERS(USER_ID),
   SOFT_ID                CHAR( 8) NOT NULL
       REFERENCES HL.SOFTWARE(SOFT_ID),
   CONFIG_ID              CHAR( 8) NOT NULL
       REFERENCES HL.CONFIGS(CONFIG_ID),
   OWN_SHARE              CHAR( 4) NOT NULL,
      PRIMARY KEY (USER_ID, SOFT_ID, CONFIG_ID)
       CONSTRAINT PK_USERSOFTWARE)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.USER_TRAINING
  (USER_ID                CHAR( 8) NOT NULL
       REFERENCES HL.USERS(USER_ID),
   TRAIN_ID               CHAR( 8) NOT NULL
       REFERENCES HL.TRAINING(TRAIN_ID),
   UNIT_COST              DECIMAL(7,0) NOT NULL,
      PRIMARY KEY (USER_ID, TRAIN_ID)
       CONSTRAINT PK_USERTRAINING)
      IN FS_HELP;

COMMIT WORK;

CREATE UNIQUE INDEX   NDX_USERNAME ON
   HL.USERS(USER_LASTNAME, USER_FIRSTNAME);
CREATE INDEX          NDX_SOFTWARE ON
   HL.SOFTWARE(SOFT_NAME);
CREATE INDEX          NDX_EQUIPMENT ON
   HL.EQUIPMENT(EQUIP_TYPE,EQUIP_MODEL);

COMMIT WORK;
```

Figure 4 -- HLDESK Schema (Page 5 of 5)

The Schema Explained

Let's look at the schema and understand what these new sections and terms mean;

1. Comments are contained between a "/*" and a "*/".

2. The first section tells us that we are creating a data base called "HLDESK" which will have a DBECon file named "HLDESK", a DBEFile0 file named "HL000" which contains the ALLBASE system catalog for the database, and a single log file named "HL001L". The size of the system catalog and log file are given in pages. Note that the page size for the log file is in 512 byte pages whereas the page size for the all other files is 4096 bytes.

3. The next command "CREATE DBEFILESET" enables the creation of a fileset, which is a domain within SQL into which MPE files may be added. In managing large data bases, you might want to create separate groups of files based on the activity of the data or the type of data. For example, you might want to put "payroll" information in one file set and "purchasing" in another.

4. Next in the header portion of the schema is the command to create two files "HL001T" and "HL001X". The first file is defined as a table type. The second is defined as an index type. This means that the table data can be physically separated from the index data, and if desired, could be placed on separate drives.

5. The next commands adds the files to the fileset "FS_HELP".

6. The next command commits the work done thus far. If no errors have occurred, a data base will exist at this point with no user tables, but with all necessary structure and with all system tables. We have also created the MPE file structure which can contain any tables which are added to the data base.

The Schema and the Schematic

To help you understand the correlation between the schema (Figure 4) and the schematic (Figure 3), I have selected a subset of tables as shown in Figure 6. Of this subset the SOFTWARE, USER_SOFTWARE and CONFIGS tables will be discussed in detail. To help with the comparison, I have listed the associated ALLBASE table schemas, without their field size definitions, on the same page.

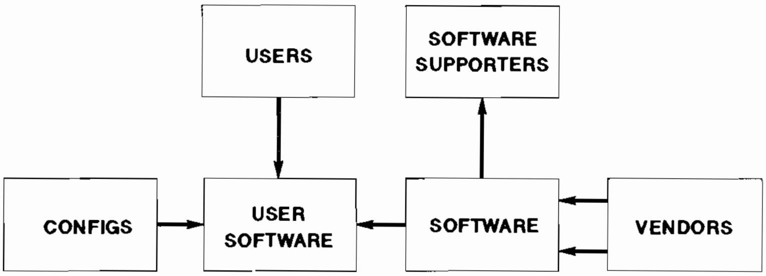


Figure 5 -- Selected ALLBASE Tables for Comparison to Schema

```
CREATE PUBLICREAD TABLE HL.USERS
 (USER_ID                    PRIMARY KEY
                              CONSTRAINT PK_USERS,
USER_LASTNAME,
  USER_FIRSTNAME,
  USER_PHONE,
  USER_LOCATION)
    IN FS_HELP;

CREATE PUBLICREAD TABLE HL.USER_SOFTWARE
 (USER_ID
     REFERENCES HL.USERS(USER_ID),
  SOFT_ID
     REFERENCES HL.SOFTWARE(SOFT_ID),
  CONFIG_ID
     REFERENCES HL.CONFIGS(CONFIG_ID),
  OWN_SHARE,
    PRIMARY KEY
    (USER_ID,SOFT_ID,CONFIG_ID)
     CONSTRAINT PK_USERSOFTWARE)
    IN FS_HELP;

CREATE PUBLICREAD TABLE HL.SOFTWARE
(SOFT_ID       PRIMARY KEY
           CONSTRAINT PK_SOFTWARE,
  SOFT_CLASS,
  SOFT_NAME,
  SOFT_RELEASE,
  SERIAL_NUM,
  VENDOR_MFG,
  VENDOR_PUR,
  DATE_PUR,
  PO_NUM,
  UNIT_COST,
  SOFT_STATUS,
  DATE_STATUS)
    IN FS_HELP;
```

```
CREATE PUBLICREAD TABLE HL.CONFIGS
  (CONFIG_ID            PRIMARY KEY
                         CONSTRAINT PK_CONFIGS,
  CONFIG_DESC,
  CONFIG_STATUS,
  DATE_STATUS)
     IN FS_HELP;

CREATE PUBLICREAD TABLE HL.VENDORS
(VEND_ID       PRIMARY KEY
           CONSTRAINT PK_VENDORS,
  VEND_NAME,
  VEND_STREET,
  VEND_CITY,
  VEND_ST,
  VEND_ZIP
  VEND_PHONE,
  VEND_CONTACT)
      IN FS_HELP;

CREATE PUBLICREAD TABLE HL.SOFTWARE_SUPPORTERS
          (SOFT_ID
             REFERENCES HL.SOFTWARE(SOFT_ID),
           SUPR_ID
             REFERENCES HL.SUPPORTERS(SUPR_ID),
           VEND_ID
             REFERENCES HL.VENDORS(VEND_ID),
           CONTRACT_ID
           DATE_CONTRACT,
           DATE_TERM ,
           TIME_AVAIL ,
           CONTRACT_COST
             PRIMARY KEY (SOFT_ID,  SUPR_ID)
              CONSTRAINT PK_SOFTSUPPORTERS)
              IN FS_HELP;
```

Figure 6 -- Selected ALLBASE Tables for Comparison to the Schematic

The two lines in Figure 5 from VENDORS to SOFTWARE are represented by the statements "REFERENCES HL.VENDORS(VEND_ID)" in the field definition for VENDOR_MFG and VENDOR_PUR. This means that one and only one entry can exist in the VENDORS table for the value of VENDOR_MFG. Zero to many records can exist in the SOFTWARE table on VENDOR_MFG for any value of VEND_ID in VENDORS. The same is true of VENDOR_PUR in SOFTWARE. Note also that the value of VENDOR_MFG can be the same, or different, than the value of VENDOR_PUR.

Note also that the names of the columns VENDOR_MFG and VENDOR_PUR do not have to match the name VEND_ID of the column in VENDORS. The size and the type of each field do have to be identical. It is also important to note that each value in the VEND_ID column in the VENDORS table has to be unique. This is denoted in the VENDORS table by the "PRIMARY KEY" association with the VEND_ID.

Note also that by looking at the table SOFTWARE, you cannot determine what tables, if any, may be referencing it . You can see that the table USER_SOFTWARE references it by the statement "REFERENCES HL.SOFTWARE(SOFT_ID)". In this case, the field name is identical in both tables. In a similar fashion, USER_SOFTWARE references USERS on USER_ID in both tables. The table USER_SOFTWARE also references CONFIGS on the column CONFIG_ID in both tables. Finally, when we look at the CONFIGS table, you see that it has no references to other tables. Looking back as Figure 3, you see that the table CONTROL_DATA has no references to other tables and that no tables reference it.

Hopefully, you can now appreciate how simply a schematic can denote a multitude of structure references and is much easier to understand than the schema. You have seen that a table:

can be standalone,
can reference a single table,
can reference a table more than once,
can reference more than one table.

You have also seen that the names for columns in the referenced and referencing tables do not have to be the same. You can also see that the primary key can be composed of multiple fields, as is the case in USER_SOFTWARE. Although not shown in this example, a table can reference another table using multiple fields as defined by a "FOREIGN KEY" (See the SQL Reference Manual).

The final piece of the schema contains the index definitions which demonstrate that in addition to the B-Tree indexes created for the primary keys, you can create specific indexes, unique or not, to expedite retrieval of data. Such indexes can significantly improve performance where large amounts of data are involved. If the amount of data is small, say a few pages, then a serial scan might be faster.

Data Clean-Up

Let's assume that on the TurboIMAGE system to be converted you have the following two sets which are relevant to the new HLDESK data base to be loaded:

```
NAME: SOFTWARE, D(1/1);
ENTRY:
  SOFTWARE-DESC,          << X40>>
  USER,                   << X4 >>
  DATE_PUR;               << I2 >>
CAPACITY:  1000;
```

```
NAME:  TROUBLE-HIST, D(1/1);
ENTRY:
  TROUBLE-ID,             << I1 >>
  SOFTWARE_NAME,          << X30>>
  USER-NAME,              << X32>>
  DATE-TROUBLE,           << X6 >>
  TROUBLE-DESC;           <<8X72>>
CAPACITY:  5000;
```

Upon inspecting the above data, you find that the software description field has both a name and sometimes a type and release number such as "Word Perfect 5.1" or "Word for Windows 2.0". You also find entries for "W/P 5.1" , "WP5.1" and "Word Purrfect". The user-id is different between the two tables, and you have entries for "George B. Scott", "George N. Scott" and "George Scott" as user names. You have comment narrative in which the comment lines (trouble-desc) may have data in the third line with the first two lines blank. You discover software names such as "Spreadsheet" and "Word Processor"...

By now you should begin to enjoy the concept that loading a data base from another electronic source more than likely will involve some form of data clean-up. In fact, the data clean-up effort may be greater than the programming effort! It is always worth considering whether re-entry of data from an original source, such as PO's or Invoices, may be simpler, faster and cheaper than cleaning up extremely dirty data.

I have found that the following conditions need to be considered as to impact on the project for typical data clean-up activities:

1. Data type/size is different.
   (If the source is longer, do you abbreviate or truncate?)

2. Data field is missing, or has blanks or zeros, and a valid value must be used.
   (Is a valid default available, or can it be derived, or does it have to be assigned?)

3. Multiple fields are in a single source field, such as full name.
   (How long to write a usable algorithm to split into tokens?)

4. Position in source is not constant such as "   Ralph Smith" and "Bob Brown" and "Jones, Jerry".
   (What is the extent of manual effort required to fix it?)

5. Field content is unknown. Field is supposed to be shoe size and contains (X, 7, and J)
   (How long will it take to find out if this is some code, or where the real shoe size is, if it exists at all?)

6. Different spellings for the same thing such as "WordPerfect" and "Word Purrfect".
   (Are there a few common variations which could be coded, or do thousands of examples exist?)

Loading the Data

My first suggestion is to (1) believe that you will load the data many times and, (2) believe that you will make data base changes as the project progresses. Working with these premises, you might consider writing a skeleton program, which can be modified with relative ease, to be able to load any table in you data base. A program like this can often be modified, tested and installed in less than one hour.

Two basic approaches exist for adding data programmatically to an ALLBASE data base; one is to use the simple INSERT verb, the other is to use the BULK INSERT verb. The simple INSERT loads one row at a time, whereas the BULK insert loads multiple rows per use. If an error is detected in a BULK insert, all rows which were being added are rolled back. If an error is detected in a simple INSERT, then only a single row is rejected. The BULK INSERT is faster, but requires that your data is error free. I highly recommend this approach for simple master type referential tables with a limited number of entries, even up to 5000 entries. If you are loading tens or hundreds of thousands of entries, the single INSERT is probably easier to manage with regard to error messages and audit trails.

Under any condition, write explicit error handling files and messages so you can quickly determine ***which*** records are being rejected and ***why*** they are being rejected.

When loading multiple tables from the same source record, I suggest that if all of the target tables cannot be loaded, then the transaction should be rolled backed so that none are loaded. This usually enables a simpler re-processing of the record after the rejection cause has been corrected.

The PreProcessor

One of the safeguards built into the ALLBASE product is that each section of code that accesses the SQL environment is added to a system table (SECTIONS). The SQL processor is sufficiently intelligent to determine that when a piece of the data base is changed, any sections using it are flagged as corrupt. Thus those programs have to be re pre-processed to insure that they are compatible with the new ALLBASE structure. Although this may seem to be a real drain on time at first, you will come to appreciate the fact that this pre-processor detects errors that the COBOL compiler could not, and will in fact save you much pain and agony in the long run.

How Big is Big Enough?

One of the questions you need to address is just how big to make your tables. Note that tables can be added to filesets if you run out of space, so in the beginning you can be conservative. Besides, you need to know how to add tables. In the ALLBASE reference manual are algorithms to help you estimate the total size of you data base. I found that these are usually within 30 % of actual and almost always within 50% of the actual (HP's numbers are usually larger than actual). If you have really big tables to load, I suggest that you load a representative sample, such as 1000 entries and get a measurement of the space before and after the load, as follows:

1. Before loading, using ISQL set all statistics for table sizes as follows:

```
UPDATE STATISTICS FOR TABLE HL.CONFIGS;
UPDATE STATISTICS FOR TABLE HL.CONTROL_DATA;
UPDATE STATISTICS FOR TABLE HL.EQUIPMENT;
UPDATE STATISTICS FOR TABLE HL.EQUIPMENT_SUPPORTERS;
UPDATE STATISTICS FOR TABLE HL.HELP_REQUESTS;
UPDATE STATISTICS FOR TABLE HL.NARRATIVE;
UPDATE STATISTICS FOR TABLE HL.RESPONSES;
UPDATE STATISTICS FOR TABLE HL.SOFTWARE;
UPDATE STATISTICS FOR TABLE HL.SOFTWARE_SUPPORTERS;
UPDATE STATISTICS FOR TABLE HL.SUPPORTERS;
UPDATE STATISTICS FOR TABLE HL.SUPPORT_REQUESTS;
UPDATE STATISTICS FOR TABLE HL.TRAINERS;
UPDATE STATISTICS FOR TABLE HL.TRAINING;
UPDATE STATISTICS FOR TABLE HL.USERS;
UPDATE STATISTICS FOR TABLE HL.USER_EQUIPMENT;
UPDATE STATISTICS FOR TABLE HL.USER_SOFTWARE;
UPDATE STATISTICS FOR TABLE HL.USER_TRAINING;
UPDATE STATISTICS FOR TABLE HL.VENDORS;
UPDATE STATISTICS FOR TABLE SYSTEM.DBEFILE;
UPDATE STATISTICS FOR TABLE SYSTEM.DBEFILESET;
UPDATE STATISTICS FOR TABLE SYSTEM.COLUMN;
UPDATE STATISTICS FOR TABLE SYSTEM.INDEX;
UPDATE STATISTICS FOR TABLE SYSTEM.TABLE;
```

2. Using ISQL, report the sizes before the test load as follows:

   SELECT DBEFILESET, NAME, NPAGES, NROWS, AVGLEN
   FROM SYSTEM.TABLE ORDER BY DBEFILESET, NAME;

3. The result should look like the following (I removed the system tables):

```
SELECT NAME,NPAGES,NROWS,AVGLEN,DBEFILESET FROM SYSTEM.TABLE ORDER BY DBEFILESET,NAME;
--------------------+-----------+-----------+-----------+--------------------
NAME                |NPAGES     |NROWS      |AVGLEN     |DBEFILESET
--------------------+-----------+-----------+-----------+--------------------
CONFIGS             |          0|          0|          0|FS_HELP
CONTROL_DATA        |          0|          0|          0|FS_HELP
EQUIPMENT           |          0|          0|          0|FS_HELP
EQUIPMENT_SUPPORTERS|          0|          0|          0|FS_HELP
HELP_REQUESTS       |          0|          0|          0|FS_HELP
NARRATIVE           |          0|          0|          0|FS_HELP
RESPONSES           |          0|          0|          0|FS_HELP
SOFTWARE            |          0|          0|          0|FS_HELP
SOFTWARE_SUPPORTERS |          0|          0|          0|FS_HELP
SUPPORTERS          |          0|          0|          0|FS_HELP
SUPPORT_REQUESTS    |          0|          0|          0|FS_HELP
TRAINERS            |          0|          0|          0|FS_HELP
TRAINING            |          0|          0|          0|FS_HELP
USERS               |          0|          0|          0|FS_HELP
USER_EQUIPMENT      |          0|          0|          0|FS_HELP
USER_SOFTWARE       |          0|          0|          0|FS_HELP
USER_TRAINING       |          0|          0|          0|FS_HELP
VENDORS             |          0|          0|          0|FS_HELP
```

4. Load the test data and repeat steps 1 through 3. The after results may look like this(I have inserted asterisks in the vendors table entry).

```
SELECT NAME,NPAGES,NROWS,AVGLEN,DBEFILESET FROM SYSTEM.TABLE ORDER BY DBEFILESET,NAME;
--------------------+-----------+-----------+-----------+--------------------
NAME                |NPAGES     |NROWS      |AVGLEN     |DBEFILESET
--------------------+-----------+-----------+-----------+--------------------
CONFIGS             |          0|          0|          0|FS_HELP
CONTROL_DATA        |          0|          0|          0|FS_HELP
EQUIPMENT           |          0|          0|          0|FS_HELP
EQUIPMENT_SUPPORTERS|          0|          0|          0|FS_HELP
HELP_REQUESTS       |          0|          0|          0|FS_HELP
NARRATIVE           |          0|          0|          0|FS_HELP
RESPONSES           |          0|          0|          0|FS_HELP
SOFTWARE            |          0|          0|          0|FS_HELP
SOFTWARE_SUPPORTERS |          0|          0|          0|FS_HELP
SUPPORTERS          |          0|          0|          0|FS_HELP
SUPPORT_REQUESTS    |          0|          0|          0|FS_HELP
TRAINERS            |          0|          0|          0|FS_HELP
TRAINING            |          0|          0|          0|FS_HELP
USERS               |          0|          0|          0|FS_HELP
USER_EQUIPMENT      |          0|          0|          0|FS_HELP
USER_SOFTWARE       |          0|          0|          0|FS_HELP
USER_TRAINING       |          0|          0|          0|FS_HELP
VENDORS             |*        38|*      1000|*       144|FS_HELP
ACCOUNT             |          0|          0|         40|SYSTEM
CALL                |          0|          0|         52|SYSTEM
COLAUTH             |          0|          0|          0|SYSTEM
COLDEFAULT          |          0|          0|          0|SYSTEM
COLUMN              |         19|        446|        160|SYSTEM
CONSTRAINT          |          0|          0|          0|SYSTEM
CONSTRAINTCOL       |          0|          0|          0|SYSTEM
CONSTRAINTINDEX     |          3|         57|        196|SYSTEM
COUNTER             |          0|          0|         32|SYSTEM
DBEFILE             |          1|          3|        136|SYSTEM
DBEFILESET          |          1|          2|         76|SYSTEM
GROUP               |          0|          0|          0|SYSTEM
HASH                |          3|         57|        196|SYSTEM
INDEX               |          3|         57|        196|SYSTEM
MODAUTH             |          0|          0|          0|SYSTEM
SECTION             |          0|          0|          0|SYSTEM
SPECAUTH            |          0|          0|          0|SYSTEM
TABAUTH             |          0|          0|          0|SYSTEM
TABLE               |          3|         58|        156|SYSTEM
TEMPSPACE           |          1|          3|        136|SYSTEM
TRANSACTION         |          0|          0|         32|SYSTEM
USER                |          0|          0|         24|SYSTEM
VIEWDEF             |          0|          0|          0|SYSTEM
```

You might also like to inspect how much total room is in the filesets using the following instructions within ISQL:

```
        SELECT  DBEFSNAME, DBEFSNPAGES, DBEFSUPAGES
        FROM SYSTEM.DBEFILESET;
```

The results might look like this:

```
        select dbefsname,dbefsnpages,dbefsupages from system.dbefileset;
        -------------------+-----------+-----------
        DBEFSNAME          |DBEFSNPAGES|DBEFSUPAGES
        -------------------+-----------+-----------
        SYSTEM             |       1000|         59
        FS_HELP            |       6000|         43
```

You might also be interested in determining how full each of the files are, as follows:

```
        SELECT DBEFNAME, DBEFNPAGES, DBEFUPAGES
        FROM SYSTEM.DBEFILE.
```

After the load of 1000 records, the results might look like this:

```
select dbefname,dbefnpages,dbefupages from system.dbefile;
-------------------+-----------+-----------
DBEFNAME           |DBEFNPAGES |DBEFUPAGES
-------------------+-----------+-----------
HL                 |       1000|         60
HL001T             |       2000|         38
HL001X             |       4000|         60
```

Note: Before the load, the three files were 59, 0, and 43 used pages respectively. So the 1000 vendor records used 38 pages of table space and 17 pages of index space. Thus if we had 100,000 vendors to add, we would have a good grasp on the total size required for the index and table space.

SUMMARY

In this paper, the following have been discussed:

1. Selecting the tools for project administration and the need for a schematic drawing tool.

2. Designing the data base.

3. Referential Integrity, Primary Keys and Indexes.

4. Building the ALLBASE Schema.

5. Explaining the Parts of the ALLBASE Schema.

6. Contrasting the Schema to the Schematic.

7. Data Clean-Up Issues.

8. Data Load Techniques (BULK INSERT, INSERT)

9. Some COBOL Constructs.

10. The PreProcessor

11. Sizing the Data Base

The presentation of a help desk oriented schema obviously was not meant to be a design for a help desk system, which is much more complex than the design presented herein. In like manner, the normalization presented in this paper was not what might be done for a large system help desk. The help desk was merely selected as a straw horse for presenting examples and highlighting points of interest.

Additional issues which are relevant to your site are security and recovery, both of which merit a separate paper. Perhaps next year, you can tell us of your experiences and knowledge in these areas.

PAPER NO.: 3153

TITLE: ALLBASE/SQL vs Image for Application Development

AUTHOR: Nicholas J. Walsh, Jr.
Speedware Corp.
8300 Boone Blvd
Suite 500
Vienna, VA 22182
(703) 848-9238

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

INTEREX '92 - Paper #3154

Early Returns on Critical Item Update: You Be The Judge...

By
Jerry Fochtman

Exxon Chemical Company
5000 Bayway Dr.
Baytown, Texas 77522
Voice: 713/425-5957
FAX: 713/425-1017

Introduction

Many of you may not have been associated with the HP user community when the Critical Item Update (CIU) enhancement request was introduced over a decade ago. However, if you've been paying any attention to the various HP user community publications or attended any user group meetings you should have heard of the CIU enhancement to TurboIMAGE by now. Although initially there was some user controversy surrounding this enhancement, Hewlett-Packard committed to implementing this long-awaited feature to TurboIMAGE at the Boston Interex Conference in 1990.

As a part of SIGIMAGE's team work with HP's IMAGE Lab, members of the SIG's Technical Review Committee were asked if there was any interest to participate in the early testing of CIU. Having been a long-time supporter of this enhancement, I eagerly volunteered to participate in the testing effort. The purpose of this paper is to outline the results of our testing so you, the user, can judge the benefits of CIU for yourself.

What Is Critical Item Update

First, let's briefly review what CIU is all about. IMAGE (TurboIMAGE) manages data as a collection of records. These records are located in either *master* or *detail* datasets. Every record in a dataset consists of one or more *data fields*. Some of these data fields may also be used to manage the storage and retrieval of records in datasets. IMAGE identifies data fields used for this purpose as *special* fields.

The nature of master datasets requires that each record be uniquely identified. The special field provided by IMAGE for this purpose is called a *key field*.

Detail datasets have no such requirement. However, the database designer may choose to designate a field on the records of a detail dataset as a special field called a *search field*.

By doing so, applications may subsequently retrieve all records having the same *search field* value without having to scan every record in the dataset.

For each such *search field*, the designer may also designate another special field called a *sort field.* If this is done, all of the records with a given *search field* value can be automatically retrieved in the order determined by the values of the associated *sort field.*

IMAGE has three intrinsics which are used to insert, remove or update records in datasets. These intrinsics are *DBPUT, DBDELETE* and *DBUPDATE.* Up to now, only the intrinsics DBPUT and DBDELETE were allowed by IMAGE rules to manage any of the *special* fields in master and detail datasets. The new Critical Item Update enhancement will now change this fundamental IMAGE rule, *but only for detail datasets.* With the new CIU feature, it will now be possible to update detail dataset search or sort fields through the use of the DBUPDATE intrinsic without having to perform a DBDELETE/DBPUT pair on the entire record.

If you would like to learn more on what CIU is all about and how you can benefit from it, I highly recommend reading Steven Cooper's paper from the 1991 Interex San Diego Conference.

CIU Implementation

(Note: The new CIU feature of IMAGE will be in the 4.0 release of MPE/iX. The version of CIU for MPE-V based systems is currently under development.)

One of the primary concerns during the investigation, development and implementation of CIU was to ensure that the new feature of the DBUPDATE intrinsic was completely compatible with existing applications. Although many people were involved with the HP's IMAGE Lab in studying the implementation proposal, when we started the actual testing with an application, several additional compatibility issues previously not considered were uncovered. To their credit, HP's Lab Team was able to quickly implement several of the recommendations from our testing without jeopardizing the release schedule.

Controlling CIU

By default, Critical Item Update will not be enabled for a database. To take advantage of CIU, it is necessary to use *DBUTIL* to enable it via the *SET* command:

```
>>SET database[/maintword] CIUPDATE = [DISALLOWED]
                                      [ALLOWED   ]
                                      [ON        ]
```

There will be three settings for CIU. **DISALLOWED** prevents any process from using Critical Item Update on the database (default). The **ALLOWED** setting indicates that a program can enable CIU using DBCONTROL. Other programs which do not explicitly enable CIU are prevented from using the new CIU feature of DBUPDATE. The **ON** setting permits any program to update critical items in the database unless the program explicitly disables this feature using DBCONTROL.

Dynamic Control

If a database setting for CIU is either ALLOWED or ON, it will be possibie for a process to use the DBCONTROL intrinsic to enable or disable the CIU feature for a specific DBOPEN base-ID. Two new modes have been added to the DBCONTROL intrinsic for this purpose. *Mode 5* will enable the updating of critical items via the DBUPDATE intrinsic. *Mode 6* will disable updating of critical items via DBUPDATE.

Determining CIU Setting

To provide information as to the current setting of CIU for a DBOPEN base-ID, the DBINFO intrinsic has been enhanced to return setting information to the calling process. Using the new option, *mode 502*, DBINFO will return to the calling process two 16-bit values. The first 16-bit value indicates the setting for the database. Possible values include:

First 16-bit value:	0 = Critical Item Update is DISALLOWED (default)
	1 = Critical Item Update is ALLOWED
	2 = Critical Item Update is ON

The second 16-bit value provides information pertaining to the current setting for the particular DBOPEN base-ID:

Second 16-bit value:	0 = Critical Item Update is disabled for this base-ID.
	1 = Critical Item Update is enabled for this base-ID.

If the first 16-bit value indicates that the database is disallowed for CIU, then the second 16-bit value will always be zero. Individual programs can utilize DBCONTROL modes 5 and 6 to affect the second 16-bit value when the database setting for CIU is ALLOWED or ON.

New IMAGE Error Messages

The new Critical Item Update feature includes several new error messages along with a change in the text for an existing error message. The revised message is for error code 41 returned from DBUPDATE. If CIU is not enabled for the specific base-ID and a call to DBUPDATE attempts to change a search or sort field, the normal error code 41 is returned. However, the error text returned for error code 41 from DBEXPLAIN or DBERROR will now be:

"DBUPDATE attempted to modify value of critical item--key, search or sort."

The change in the text for this message attempts to outline the possible causes rather than identify a specific reason for the error. It will be necessary for the user to identify the specific dataset involved to determine if the error is related to a *key field* for a master dataset or a *search* or *sort* item associated with a detail dataset.

When CIU is enabled and the process attempts to change a search or sort field, many of the same types of errors could occur with DBUPDATE as with DBPUT. When these exception conditions do occur, DBUPDATE will continue to return an error code 41 in the first status word. However, additional error information will be contained in the third 16-bit word of the status array. The list of possible values are:

Third Status Value:	1xx - No chain head for master entry for path xx.
	2xx - Full chain for path xx (contains 2,147,483,647 entries)
	3xx - Full automatic master for path xx.
	4xx - Full automatic master synonym chain for path xx.

Special Considerations

There are two application design methods which will impact your ability to take advantage of the new CIUPDATE feature of IMAGE. It is important to ensure your applications are not using these techniques before proceeding.

The first consideration relates to the technique of applications relying on DBUPDATE not being able to change search or sort fields. In most cases this was a security control mechanism. However, with the new CIU feature, the ability of a program to update these search or sort fields with DBUPDATE may present problems for these applications. In this case, do not set CIU to the ON setting for the application's database(s). Instead, consider setting CIU for the database to the ALLOWED mode. This would allow those programs which do not rely on error 41 from DBUPDATE to take advantage of CIU by enabling this feature using the new DBCONTROL modes. All other programs which rely on error code 41 from DBUPDATE would continue to function as before.

The second consideration relates toapplications whic perform DBPUTs and use n item listthat does not contain all the data fields in the detail dataset record. One of the biggest advantages of CIU is the ability to add additional search paths to detail datasets without having to change application programs. Doing this will greatly facilitate the use of reporting tools to accommodate changing reporting needs without requiring application modifications. Unfortunately, if incomplete item lists are used in calls to DBPUT and one of the missing items becomes a new search field, all subsequent calls to DBPUT will fail with error code -53: "Missing search or sort item". This is because DBPUT requires that all search fields be included in the item list. In this instance it will still be possible to utilize CIU. However, in order to achieve the increased flexibility of adding additional detail dataset search paths possible with CIU, the application software would require modification.

CIU Testing

One of the first tasks in arranging the testing was to identify a site where we could conduct the testing without impacting their normal operations. As you can imagine, most companies do not have "spare" computers which can be used as crash-and-burn test systems. Fortunately for us in the HP environment, we have companies like Adager who generously offered the use of their facilities and resources for this testing. In preparation for the testing, Rene Woc of Adager worked with HP's IMAGE Labs in obtaining and installing the necessary test software along with configuring disc capacities for our anticipated testing needs.

At the same time I received information from Jim Sartain, HP's IMAGE R & D Project Manager, on the testing HP was conducting so we would not duplicate our efforts. Given this information and several conversations with others, I was able to develop several test scenarios for validating CIU. In addition, there was an extensive application I was working on which was coded in a manner that would take advantage of CIU when the enhancement was available. This too, would be used in testing CIU.

Armed with this data, I, along with the talents of Wirt Atmar, Fred White and Rene Woc set out on an effort to "break" CIU. As a team, we spent four days together, discussing the intricacies of CIU and testing areas where we thought there may exist a window of vulnerability caused by the enhancement. Each test scenario was carefully orchestrated to ensure the results were an accurate reflection of the conditions being tested.

Test Environment

Our testing of CIU was conducted on an HP-3000 Series 922 which had 32 MB of memory along with 4 - 670MB disc drives. The CIU enhancement required version 3.0 of the MPE/XL operating system. Use of the system was restricted to the CIU testing effort.

The database used for our tests was simple in design. It consisted of a single detail dataset with one or more automatic masters attached to it providing search paths as needed for each particular test. The ADAGER database tool was utilized to add or remove search or sort fields as necessary for the tests. All fields were 4-character ASCII fields. We did not have enough time to conduct the same tests using numeric search or sort fields to assess any impacts they may introduce. The general feeling among the team was the specific field data type would not impact CIU.

CIU Testing Process

CIU testing was conducted in three phases. The first phase focused on validating that CIU did not introduce any problems with IMAGE's integrity. The second phase involved assessing the performance impact of CIU. The final phase involved enabling CIU for an existing application to assess any compatibility problems.

Phase 1: Testing CIU Functionality

In the first phase, a total of ten different tests were conducted. The following are the tests that were conducted and their results:

Test 1

Description: Using a single program which opens a database twice, simulate 2 users using different detail search paths to access the same detail dataset. Both user A and user B retrieved the same record via different paths. Once user B had retrieved the record, User A did a dataset lock, changed the field on the record associated with user B's path, performed an update of that record and then unlocked the dataset before user B performed a DBGET mode 5 (ie. chained get next).

Findings: As expected, when user B executed the DBGET next, an error condition 18 - broken chain, was returned. This was caused by the backward pointer of the next record not matching the record number of the last record read.

Conclusion: This type of erroneous broken chain error has always existed when applications do not lock when reading or else use incompatible locking strategies. The IMAGE manual documents the potential of this erroneous error when inadequate locking strategies are used. The test shows that CIU has not introduced any additional factors into this known phenomenon.

Test 2

Description: Two separate programs were executed on different terminals accessing the same detail dataset using different paths. User A would perform an item lock, find and get a record, pause for terminal input, retrieve the next record and then perform an unlock. While user A was waiting for terminal input, user B would perform an item lock on a different detail search path, find, get, change the key on its path followed by an update and unlock. The record being changed by user B was physically in the same block as user A's read. The values of the keys in this test were such that the users were working on different records which were not on each other's search path.

Findings: When user B attempted to perform the update, the user was held off, apparently by IMAGE, until user A completed the terminal input and moved forward, performing the unlock. Once user A did the unlock, user B's update proceeded.

Conclusion: When users are operating on completely different records within the same IMAGE block, IMAGE correctly single-threads the users in order to maintain database integrity.

Test 3

Description:	This test was similar to test 1 except that instead of changing the record just read by user B, user A obtains the next record to be read by user B, changes it so it would no longer be in the path currently being read by user B. This case simulates a forward pointer problem vs. a backward pointer problem as in test 1.
Findings:	User B received a broken chain error (error code 18) on the DBGET for the next record in the chain after it was changed to a different search field value by user A.
Conclusion:	Same as test 1, CIU did not introduce any new problems in this set of circumstances. The erroneous broken chain error was expected due to an inadequate locking strategy. The integrity of the database remained intact.

Test 4

Description:	This test involved updating the value of a sort field.
Findings:	CIU correctly updated the record, changing the position of the record within the sorted path without physically relocating the record in the database.
Conclusions:	Functions as intended.

Test 5

Description:	This test was designed to verify CIU's detection of a full automatic master dataset and return an error code 41 with the appropriate extended error description. In this test, entries were added to a detail dataset using 3 different search values. This filled up the related automatic master whose capacity was 3. We then attempted to update the search value of a detail entry on a multi-entry chain with a value different than any of the 3 existing values.
Findings:	CIU returned an error 41 with an extended error value indicating that the automatic master on the particular path was full. The record we were attempting to change remained in its original state.
Conclusion:	CIU correctly detected full master conditions and provided the additional information to clarify the update failure with no impact on database integrity.

Test 6

Description: This test was a variation on test 5. The update was on the only entry of a single-entry chain. This would determine whether or not CIU examined the ramification of the change before determining if any capacity problems existed.

Findings: The attempt to change the single record search value to a new search value not previously used was successful.

Conclusion: CIU indeed considered the affect of the whole transaction on automatic master before concluding any capacity problem existed.

Test 7

Description: This test involved setting up a single automatic master which had two paths to the same detail dataset. This type of arrangement may occur when using a single master to contain dates and one path to the detail dataset may be an order date and another path may be the date of shipment. Test data was generated such that the automatic master was completely full. We then attempted to change the second search field on a detail record to a value not already in the automatic master.

Findings: CIU returned an error 41 with the extended condition which indicated a full master dataset. However, the search path identified in the extended error data indicated the condition was related to path 1 instead of path 2. In working with HP's IMAGE Lab, together we identified that the problem was not with the CIU enhancement, but rather the methods we were using to manage the testing conditions. After correcting our test conditions we reran the test and received the correct path identification in the extended error information.

Conclusion: Functioned as intended, with appropriate indication as to the search path associated with the error condition.

Test 8

Description: This test was a combination of tests 6 and 7. As in test 7, a single master served 2 separate search paths to a detail dataset. Again, we constructed test data such that there was only one detail record with a particular date. However, the date was in both search fields on the single record. The test involved changing both search fields to the same value which was not already in the master. As before, the automatic master was at 100% capacity.

Findings: As in test 6, CIU examined the whole transaction and realized that the change would have removed an entry from the master thereby making room for the new entry. On this basis the transaction was allowed to proceed.

Conclusion: Functions as intended.

Test 9

Description: As a variation on test 8, the second date field on a single detail record was the only occurrence of the particular date. The test changed this date to a new date which did not previously exist in the full master.

Findings: As in the previous tests, CIU examined the results of the whole transaction and properly allowed the transaction to proceed.

Conclusion: Functions as intended.

Test 10

Description: This test was to determine if CIU took advantage of knowing its current location in a sort chain when changing the sort field value so as not to have to traverse the entire sort chain looking for the new position. To conduct this test, a large sort chain of over 10,000 entries was constructed. The first 4,998 records had sort values of "AAAA", the next three records had sort values of "LLLL", "MMMM" and "PPPP" and the remaining records had sort values of "ZZZZ". Between the second and third "ZZZZ" records following the "PPPP" value we introduced a broken chain by using ADAGER's super-user "bits and bytes" feature to turn off the active record flag. When DBPUT adds a record to a sorted chain it starts at the last entry of the chain and reads the chain backwards to locate the position in the sort to link the new entry. By introducing a broken chain, we would determine if CIU also starts at the last entry of the chain or took advantage of knowing its current position in the chain when updating a sort field. In addition, PROCTIME calls were inserted around the DBUPDATE to measure the amount of time used to perform the update. Given the large number of "ZZZZ" entries, if DBUPDATE was functioning like DBPUT and starting from the last entry in the chain, the amount of processing time would be greater than if DBUPDATE was searching for the new location based upon the current location.

Findings: When changing the single record with the sort value of "MMMM" to a value of "HHHH" we did not receive any broken chain errors. Furthermore, the amount of processing time was extremely small as compared to the amount of processing time to traverse 5,000 entries in a chain. Other variations of the sort value which were above "ZZZZ" in the sort sequence provided similar results.

Conclusions: The initial test findings indicate that CIU uses it knowledge of a record's current position in a sort chain to move an entry to a position in the sort path when the new value of the sort field is less than the previous value. This particular test is inconclusive in terms of assessing whether or not CIU uses this same knowledge and moves forward from the entry's current position when the new sort field value is greater than the previous value. Furthermore, additional testing would be needed to assess whether or not CIU examines the active entry flag or simply traverses the chain pointers. While not conclusive, this particular test case suggests that CIU may be much more efficient in updating sort fields than using DBDELETE/DBPUT. (CIU may breathe new life into the use of sorted paths as it relates to performance!)

Conclusions From CIU Functional Testing

There were many other scenarios which we wanted to test. However, given the limited time, we needed to proceed to our next phase of testing. From these general tests we concluded that CIU was indeed functioning properly within the rules of IMAGE and had not affected the integrity of IMAGE databases. Despite our collective efforts, we were unable to break CIU!

Phase 2: Testing CIU's Performance

The second phase of testing involved examining the performance of CIU against the traditional methods of changing detail dataset search and sort field values. The performance testing was divided into 2 parts. The first part focused on the timings of DBDELETE/DBPUTS versus CIU when changing search fields whose paths were not sorted. There were 2 cases that were studied; one involved updating a single search field; the other case involved updating all search fields. The second part of the performance testing involved sorted paths.

Performance Testing: Part 1, Case 1

This series of tests involved comparing the performance of CIU against a DBDELETE/DBPUT pair when changing a single search field value on a detail entry. The number of search paths to the detail dataset was increased from 1 to 16, conducting the complete set of tests each time a new search path was added. Between each test the detail dataset was completely erased and reloaded using the same set of data records. This ensured that the data being changed and the location of the data was identical for each test. In addition, no other processes were running on the system during these timing tests.

The following steps were used to conduct the various performance tests:

Step	Description
1	Set up the database with a single search path to the detail dataset.
2	Erase the database and place 100 identical records in the detail dataset using a program.
3	Execute a program which performs DBDELETE/DBPUT, changing a single search field value to a new value which would be the same for all 100 records. Capture PROCTIME around just the DBDELETE/DBPUT pair and summarize this value for all records.
4	Perform steps 2 through 3 at least 5 times, recording the PROCTIME total from each pass.
5	Reset the database as outlined in step 2.
6	Using another program which performs DBUPDATEs, change the same single search field value to the new value as in step 3, accumulating PROCTIME around the DBUPDATE call for each of the 100 records.
7	Perform steps 5 and 6 at least 5 times, recording the PROCTIME total from each pass.
8	Using ADAGER, add an additional search path to the detail dataset and restart the testing at step 2 until a total of 16 search paths have been tested using these steps.

This testing collected a total of 180 timings; 90 from the DBDELETE/DBPUT pair and another 90 from DBUPDATE using CIU. The 5 test values for each case were averaged. The following table shows the results (unit of measurement is in milliseconds for 100 records):

No. Of Paths	Delete/Put Avg. Time	Std. Dev.	NSDD*	CIU Avg. Time	Std. Dev.	NSDD*
1	1,597.0	11.36	40.96	966.2	20.66	-0.15
2	2,062.2	31.52	12.75	963.2	8.84	-0.52
3	2,464.0	32.37	16.69	958.6	3.13	3.32
4	3,004.2	30.57	10.39	969.0	8.86	0.79
5	3,322.0	17.15	22.02	976.0	8.60	-0.53
6	3,699.6	23.90	19.12	971.4	12.52	1.47
7	4,156.6	49.95	9.51	989.8	10.69	3.84
8	4,631.8	20.08	16.74	1,030.8	27.64	-0.84
9	4,968.0	33.23	15.08	1,007.6	12.84	0.56
10	5,469.0	51.64	7.44	1,014.8	7.63	6.63
11	5,853.2	67.50	5.69	1,065.4	27.87	-2.43
12	6,237.0	56.65	7.96	997.6	6.80	0.91
13	6,688.2	61.05	5.50	1,003.8	10.66	-0.09
14	7,024.0	24.77	19.56	1,002.8	12.58	1.19
15	7,508.6	87.40	6.58	1,017.8	12.74	0.42
16	8,083.4	75.28		1,023.2	21.82	

*NSDD - Number of Standard Deviations Different

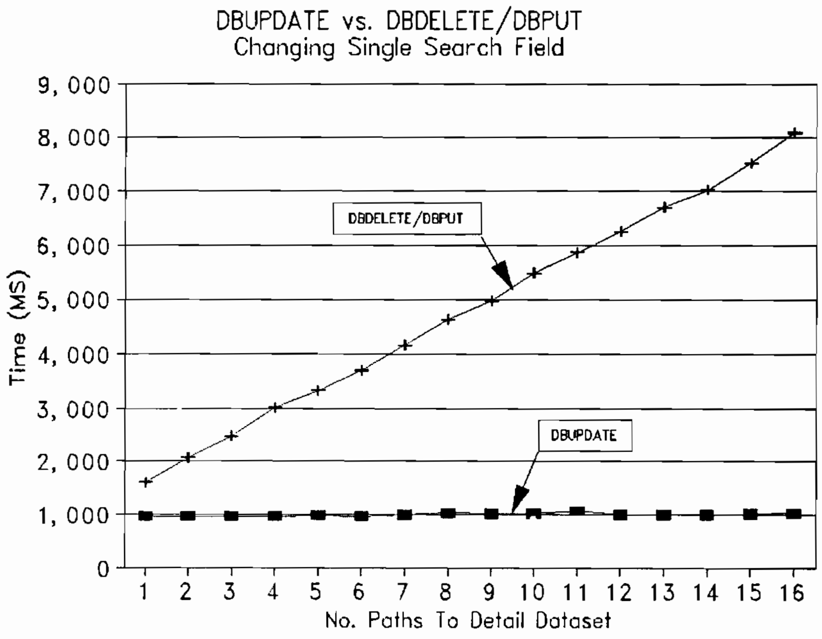
A statistical analysis of the data showed there was negligible experimental error in the samples. The analysis also showed the samples were tightly coupled about the average indicating little variation between samples for a given path count. A standard deviation was also calculated between adjacent paths (NSDD) to assess the significance of the differences between path counts.

In the case of the DBDELETE/DBPUT pair, the large standard deviation values between cases when the number of search paths increase (NSDD) indicates essentially that each case is greatly different than the next. This implies that the number of search paths to the detail dataset has a significant impact on the time to perform a DBDELETE/DBPUT, even when changing a single search value.

On the other hand, examination of the timings for CIU showed no statistical significance whether there was a single search path the detail dataset or multiple paths when changing only a single search field value. This consistency indicates that the number of search paths has no impact on the performance of CIU when updating a single search field value.

In comparing the two methods, CIU is essentially 1.5 times faster than DBDELETE/DBPUT when there is only a single path to the detail dataset given this specific test case. As more search paths exist on the detail dataset, CIU's advantage over DBDELETE/DBPUT increases by an average value of approximately 50% per record for each added search path until it performs approximately 8 times faster for 16 search paths.

The following graph illustrates the performance advantage of DBUPDATE over DBDELETE/DBPUT as a function of the number of paths to a detail dataset:



Performance Testing: Part 1, Case 2

The second case of testing CIU without sorted paths was to compare DBUPDATE against DBDELETE/DBPUT when all search paths were changed. In changing all search values for an entry, DBUPDATE would more closely resemble DBDELETE/DBPUT in terms of affecting all search paths for the entry.

The test database and the testing steps were identical to those used in the previous test case. In this instance only DBUPDATE was tested, changing all search field values for each entry. It was not necessary to gather performance data on DBDELETE/DBPUT because the test conditions for DBDELETE/DBPUT were was identical to case 1.

The following table shows the results of this test compared against the DBDELETE/DBPUT data from the previous case:

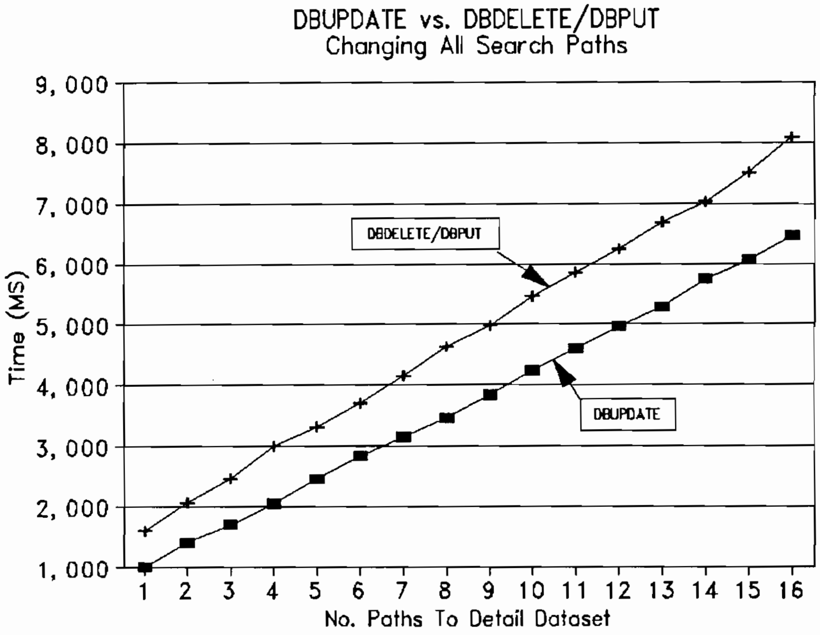
No. Of Paths	Delete/Put Avg. Time	Std. Dev.	NSDD*	CIU Avg. Time	Std. Dev.	NSDD*
1	1,597.0	11.36	40.96	1,008.6	12.30	32.57
2	2,062.2	31.52	12.75	1,409.2	9.58	30.58
3	2,464.0	32.37	16.69	1,702.0	15.92	21.76
4	3,004.2	30.57	10.39	2,048.4	19.68	20.97
5	3,322.0	17.15	22.02	2,461.0	25.66	15.00
6	3,699.6	23.90	19.12	2,846.0	20.84	14.87
7	4,156.6	49.95	9.51	3,156.0	29.77	10.54
8	4,631.8	20.08	16.74	3,469.8	17.02	21.57
9	4,968.0	33.23	15.08	3,837.0	44.02	9.05
10	5,469.0	51.64	7.44	4,235.4	27.28	13.31
11	5,853.2	67.50	5.69	4,598.4	29.67	11.98
12	6,237.0	56.65	7.96	4,953.8	17.74	18.62
13	6,688.2	61.05	5.50	5,284.2	12.24	37.29
14	7,024.0	24.77	19.56	5,740.4	34.22	9.37
15	7,508.6	87.40	6.58	6,061.2	42.19	9.28
16	8,083.4	75.28		6,452.8	64.00	

*NSDD - Number of Standard Deviations Different

The analysis of the data indicates that the performance of DBUPDATE is also impacted by the number of search paths that are changed. This is consistent with the previous results for DBDELETE/DBPUT.

However, comparing DBUPDATE to DBDELETE/DBPUT shows that the performance of DBUPDATE is consistently better than DBDELETE/DBPUT even though all search paths are changed. The overall average performance of DBUPDATE is 20-40% better than DBDELETE/DBPUT pair when changing the same number of search paths for an entry.

The following diagram illustrates this consistency:



Performance Testing Conclusions: Part 1

Based upon the specific test environment we concluded that overall, CIU offered a performance advantage over DBDELETE/DBPUT when changing detail dataset search or sort fields. In our specific case, the tests showed a 20% to almost 800% advantage. However, these results are specific to our testing conditions and not necessarily indicative to all situations. Factors such as data locality, computer performance, available memory and volume set conditions will impact your specific results. Furthermore, these test results were based upon the MPE-XL implementation of CIU. Given the differences between MPE-XL and MPE-V, it would be inappropriate to assume similar results may occur on the MPE-V release until testing is conducted to quantify the performance benefits.

Performance Testing: Part 2

The second part of performance testing focused on attempting to measure the performance of CIU vs. DBDELETE/DBPUT when changing a sort field. Our plans were to employ fairly large sorted chains for this purpose. Unfortunately we uncovered a previous problem in IMAGE unrelated to CIU that had resurfaced. Apparently there is a known problem where during the processing of large sorted chains IMAGE affects the accuracy of the millisecond processing time counter for the particular program.

In discussing our findings with HP's IMAGE Lab, they indicated that they were familiar with the problem and were working to address it. While the problem does not affect the integrity of the database, it did impact our ability to collect performance information for this portion of our testing.

Phase 3: Application Testing

The final phase of testing involved actually applying CIU to an application. For this purpose, I used an application which was designed to take advantage of CIU when it became available. It was a large COBOL application which utilized up to 18 different databases depending on what activity the user was performing.

The two special considerations previously outlined on page 3154-4 did not apply to this applications, so CIUPDATE could be set in the ON mode in each database. Once enabled for CIU, a number of normal operations were conducted without any detectable errors. The actual databases were then examined to insure the outcome of the transactions was correct. Unfortunately, attempts to measure changes in application performance were not conclusive given the nature and complexity of the application software. However, the outcome from this effort identified several implementation scenarios that had not been previously considered. These conditions were reported to the IMAGE Lab and used to improve CIU implementation.

Conclusions

The majority of this paper has focused on the potential benefits CIU offers to existing applications. One cannot over emphasize the possible benefits of CIU in terms of its flexibility to enhance the indexing of existing databases. This new flexibility has the potential of significantly accelerating data retrieval speeds in under-indexed databases, thereby enhancing the ability of IMAGE to meet changing business information needs.

Now that all the dust has settled and we have actual data on Critical Item Update, one has to ask whether the our collective effort has been worth the benefits. I believe I speak for everyone involved with the effort when I say the performance results equal or exceed our expectations. In addition, the increased indexing flexibility will greatly enhance the use of report writers. CIU frees the user from the artificial constraints, which were limiting, unnecessarily, the usefulness of IMAGE. However, the final judgement rests with those who use this new feature to solve business needs. We now await your judgement.

Special Thanks

I believe that I speak for everyone involved with this effort when I say that the team work, dedication and open, candid communication that was demonstrated between HP's IMAGE Lab, the user community and the third-party vendors was the basis for our success in implementing this enhancement. By continuing this partnership IMAGE will continue to grow and be an integral part of the HP computing environment.

Over the years there have been numerous other people involved in attempts to progress this enhancement request to IMAGE. The number of people who have been involved in one way or another is too numerous to mention and I would probably inadvertently omit several. However, there are two people who have consistently been behind this enhancement to IMAGE for many years. They are F. Alfredo Rego and Wirt Atmar. Without their persistence and support, the Critical Item Update enhancement to IMAGE would perhaps still be a pipe dream.

In gratitude....

The author wishes to express his appreciation to Jim Sartain and R.J. Diepeveen of HP's IMAGE Lab, Wirt Atmar of AICS Research, Inc, Fred White, Rene Woc and F. Alfredo Rego of Adager for taking the time to review and provide their thoughtful feedback in helping to refine this paper.

About the author.....

*Jerry Fochtman has been engineering software on HP-3000 computers since 1974. He began sharing his HP experiences with an article published in the first issue of the Hewlett-Packard Journal (Vol. 1, No. 1, May, 1977). Since that time he has written numerous papers for local publications and occasionally for some of the national and international publications for Hewlett-Packard users. In addition to his papers, Mr. Fochtman has made numerous presentations at both the local and national level. In 1989 he co-authored a book on MPE internals entitled **Compass**. His involvement with the Critical Item Update enhancement to IMAGE began with the 1985 Interex Conference in Washington, D.C..*

Bibliography

Atmar, Wirt; "The World's Oldest Enhancement Request"; Interact, August, 1990; Vol. 10, No. 8, pp. 112-118

Boyd, Larry and Beauchemin, Denys; "IMAGE Critical Items: Do you want this intrinsic?"; The HP Chronicle; November, 1990; Vol. 7, No. 12; pp. 42-44.

Cooper, Steven; "Critical Item Update - What Will It Do For Me?"; INTEREX 1991 San Diego Conference Proceedings; Vol. 1; Paper #3219.

Cooper, Steven; "IMAGE Enhancements Aren't Too Costly or Difficult"; The HP Chronicle; January, 1991; Vol. 8, No. 2; pp. 70.

Fochtman, Jerry; "Facts Make Case For Critical Item Update"; The HP Chronicle; January, 1991; Vol. 8, No. 2; pp. 36-38.

Rego, F. Alfredo; "Performance: How do I get more in both MPE/V and MPE/XL?"; INTEREX 1989 San Francisco Conference Proceedings; Vol. 1; Paper #5530.

Sanders, David; INTEREX 1990 Boston Conference Management Roundtable Questions and Answers; pp. 5.

Paper #:3155

Title : Using Oracle to Access a TurboIMAGE Database

Author : Kimble F. Ross
Company: Database Consultants Inc
4835 LBJ Freeway
Suite 900
Dallas, Tx 75244
(214) 392-0955

I. BACKGROUND

The need to share data between the VAX platforms and the HP platforms has existed here at this manufacturing site for some time. In general, Engineering data resides on the VAX system while Business and Manufacturing data resides on the HP 3000 MPEXL system. Even though the data has naturally been segregated between platforms, it has become increasingly evident that sharing data between platforms would add value and functionality to both systems. Currently, data is being shared with the existing tools. Bulk transfers of data between platforms are currently facilitated by data extracts and using FTP(file transfer protocol) to transfer data across platforms. The benefits of sharing data between platforms outweighs the considerable effort that is needed to currently share data between platforms.

The demand for data sharing has increased considerably over the past year and is predicted to continue. The Information Systems Strategic Planning study has revealed specific areas where data between the VAX and HP systems needs to be shared. Requests for software that require data that is remote to the application are being postponed or not considered because of the inability to currently share data across the VAX and HP platforms effectively. After several high level meetings between members of the Information Systems group on the subject of data sharing, it was concluded that if the process of data sharing could be simplified then more data could be shared and the cost of sharing would be reduced. Data sharing would translate into added value to our existing data and added functionality to our data processing systems. A search was conducted for platform connectivity tools between the VAX and the HP resulted in Oracle products providing a solution to sharing TurboIMAGE and Oracle data across the VAX and HP platforms. This need to share data led the Information systems group to evaluate the Oracle relational database and Oracle networking products. The document will also touch on some of the benefits and challenges of implementing relational technology in a traditional TurboIMAGE environment.

II. EVALUATION CRITERIA

The following will be the criteria used to evaluate the ORACLE relational database and its associated products. The criteria are in priority order with the most important item at the top.

Criteria 1 The database should have the ability to read and write data that resides in the ORACLE database with the HP platform being the client and the VAX platform as the server and vice versa.

The functionality in this criteria was demonstrated early on in the evaluation. The Oracle database along with the SQL*Net product provide read client server capability. This capability is available by simply installing the SQL*Net product and adding the database links. There was no maintenance required after the initial insulation. This is all handled at the database level therefore products such as Smartstar and soon to be available Speedware inherently have this capability to access a remote Oracle database.

Oracle satisfied Criteria 1 to provide client server capability between the VAX and HP platforms. The Oracle evaluation retrieved one million rows from the client machine as fast as retrieving the same set of data from the local TurboIMAGE database using the TurboIMAGE intrinsics. The test exercised VAX client to HP server and HP server to VAX client access reading and inserting and deleting remote data.

Oracle is currently working on an improved version of SQL*Net that will provide added functionality and increased performance to the networking product. Version 1 SQL*Net consumes one server process for each client process requesting data. Version 2 of SQL*Net (currently beta) will service clients utilizing a single process. Version 1 SQL*Net also requires the application to connect to the remote database to do updates, deletes or inserts. Version 2 SQL*Net will allow remote database updates, deletes and inserts.

Criteria 2 The database should have the ability to read HP TurboIMAGE data from the HP or VAX platform.

The SQL*Connect to TurboIMAGE product provides a method to access the HP TurboIMAGE database utilizing relational SQL access. This product will provide the functionality without disk space consumption or requiring additional software development to the Oracle applications on the VAX platform. The TurboIMAGE database essentially appears as an Oracle database. The Vax platform can access the TurboIMAGE database as if it were just another node in the Oracle network. Performance on the Vax was surprisingly good retrieving a one million row dataset from TurboIMAGE. SQL access of the TurboIMAGE database performance was very good and sometimes better than the current Image intrinsics accessing the TurboIMAGE database locally.

This tool can be used as a migration tool from TurboIMAGE to Oracle on the HP platform or as an application tool on the VAX platform. Software utilizing SQL access can be developed in our current TurboIMAGE environment. Data can remain in the TurboIMAGE environment or if the application requires update, delete or insert capabilities the data can be converted from the TurboIMAGE database to Oracle database. Data can be converted from the TurboIMAGE database to the Oracle database with a one line statement SQL create statement. Oracle satisfied this criteria by providing SQL access to the TurboIMAGE databases from both the VAX and HP platforms.

Criteria 3 The Oracle database must be able to integrate with our existing software.

The primary software development tools on the VAX are Smartstar and FORTRAN both of which interface very well with Oracle utilizing the SQL language. The manufacturing system on the HP platform is written in FORTRAN with some new systems written in Speedware. The most popular development language, FORTRAN, interfaces very well with the Oracle database on the HP platform. Speedware is currently enhancing their product to allow SQL access. The SQL data retrieval method is simpler than utilizing the TurboIMAGE intrinsics (see software development section below). The utilization of SQL in software development can shorten development time and reduce the software maintenance cost. The development methods used on both the VAX and HP should become more common utilizing SQL. Using SQL access will allow software to be portable between both platforms. All the software developed to test performance was easily ported from the HP platform to the VAX platform. Currently Software is not portable between the HP and Vax platforms in our current TurboIMAGE environment. Oracle satisfied this criteria by allowing the database to integrate with our existing software.

Criteria 4 The Oracle database products must be able to integrate with our existing data.

This criteria is focusing on the products that Oracle provides to do software development and the file systems they can access. The existing data that resides on the VAX platform resides in VAX RMS, Ingress, and Oracle. HP's existing data resides in TurboIMAGE, MPE's direct files, and MPE's Ksam files. Oracle software development products currently access only the Oracle Database with the exception of SQL*Connect to TurboIMAGE and the PRO*FORTRAN precompiler. The SQL*Connect to TurboIMAGE provides the ability to select TurboIMAGE data with Oracle products but there is no facility to delete, insert or update data. The Oracle precompiler gives the developer complete freedom to access any file system that FORTRAN can access which includes all the MPE file systems and the Oracle Database.

The VAX software developers chose to utilize the Smartstar product and Oracle's FORTRAN precompiler as opposed to utilizing the SQL*Forms and SQL*ReportWriter. This solution provides the major advantage of being able to access all VAX's various files systems. The combination of the FORTRAN precompiler and Speedware satisfy this criteria on the HP platform. The HP software developers can currently use the FORTRAN precompiler to access all the HP file systems and the Oracle database. This solution gives both groups a method to access all their respective file systems with either a third generation language or fourth generation language. Speedware currently accesses all the HP file system types and is developing access to the Oracle database. SQL*Forms and SQL*Reportwriter do not satisfy this criteria but the PRO*FORTRAN precompiler and Speedware will.

Criteria 5	The Oracle database must be reliable and robust.

The Oracle database and products performed reliable and were robust during the 60 day demonstration period. The Oracle database on the HP platform was intentionally left up while the HP hardware system was brought down abruptly for backups and system maintenance. This is similar to a power failure or other type of abrupt halt on the system. The Oracle database was simply brought up in a consistent state after each of the frequent halts. There is a risk that corruption may occur in a TurboIMAGE environment when the system is abruptly brought down while users are accessing the database. The corruption in the TurboIMAGE environment can require the system to be inaccessible for hours while the broken chains in the database are rebuilt or the database is restored. The risk of corruption caused by a unscheduled machine halt seems to be handled will in an Oracle environment. The Oracle database will also not allow a deadlock to the database to occur as is possible in a TurboIMAGE environment. This failsafe feature within Oracle will prevent the sudden halt of the system that is possible in a TurboIMAGE environment. The Oracle database satisfied this criteria.

The Oracle database provides many robust features not provided within the traditional TurboIMAGE environment (see Intrinsic benefits below). The Oracle database does not require additional products such as Netbase or ADAGER to provide networking capability or database change ability because this functionality is included in the Oracle database. There are many additional Oracle features that are available in the next release of Oracle version 7 (currently in beta). Oracle satisfied Criteria 5 by providing a reliable and robust database environment compared to the traditional TurboIMAGE environment.

Criteria 6 — There must be tools available to increase our development and operational productivity.

The Oracle database combined with the SQL*Net products and PRO*FORTRAN present the software developer with a superior environment to design, develop and maintain applications. This environment provides Client Server computing, partial key retrieval, simplified data access using SQL, improved locking mechanism, superior transaction control, smaller application code size, on-line data dictionary, logical views of data, application and data independence and many others described in the intrinsic benefits section of this paper.

The operational environment is also improved over the current TurboIMAGE environment. This environment provides efficient disk space consumption, improved disk space management, hot backups, simplified and on-line database changes and other benefits described in the intrinsic benefits section of this paper. Oracle satisfied Criteria 6 by providing functionality to increase development and operational productivity.

Criteria 7 — Oracle application performance should be acceptable and ORACLE applications should not impact the existing TurboIMAGE applications.

Acceptable performance will be defined as equal to TurboIMAGE performance plus or minus 20% of TurboIMAGE performance. This criteria is the last and least significant criteria so it should be weighted appropriately. The functionality of being able to share data is of greater importance than performance issues. One of the tasks during the evaluation was to develop software to exercise the Oracle database and to test the different paths and locations of data. Test software was developed to stress the Oracle database and the SQL*Net products. One of the largest sets of data on the manufacturing system containing million records was used in the testing. This development and testing of software occurred on the HP DEV machine during peak load hours. Once the software was developed it was ported to the VAX DEV machine and tested under similar conditions. The conclusion was that Oracle applications did not significantly impact the existing TurboIMAGE applications. Performance of the Oracle applications was equal and sometimes faster than the same application written in TurboIMAGE. The Oracle database and

SQL*Net products satisfied Criteria 7 by performing well and not impacting the existing TurboIMAGE applications.

III. RELATIONAL DATABASE INTRINSIC BENEFITS

Intrinsic benefits can be realized when moving from a network database such as TurboImage to a relational database such as Oracle. This section will compare and contrast TurboIMAGE with Oracle.

A. DATA SHARING

Data can be shared between the VAX and the HP platforms on a transaction basis utilizing Oracle products. Currently data is physically copied to the remote platform, or worse, it is being hand keyed into the other system. This method of copying data requires data to be extracted, copied over the network to the other system, and then collected into the other system. This process consumes man hours, additional machine resources, network resources and capital resources.

TURBOIMAGE DATA SHARING
The current method of data sharing will extract, transfer, and upload the entire set of data on a periodic basis. Since the rows of data that are needed can not be anticipated, the entire set of data is transferred. The shared data that is truly accessed may only be a small subset of the entire data set. This method of sharing data is inherently inefficient because larger volumes of data are transferred than necessary and it is transferred more frequently. The data transferred in a SQL*Net environment is low volume because only the data needed is transferred.

The current method of data transfer also creates the problem of having two sets of data. File transfers are inherently out of sync with the live system. The primary data set is where updates, inserts, and deletes occur while the remote data contains a copy or snapshot of the primary data when the data was extracted. Since the remote data set has a frozen view of the primary data set, it will become stale soon after the copy is made. As time marches on and updates, deletes or inserts occur, the primary data set and the remote data set will become more and more out of sync requiring a new copy to be made. This situation is similar to comparing a current report with a report that is a week old. The current report is fairly accurate while the old report does not reflect the actual data in the system. In an Oracle environment the data is always current. This aging of data in the current environment forces the file to be transferred more frequently to keep it as fresh as possible on the remote machine. Increasing the file transfer rate will increase man hours, machine and network resources consumed. This method of sharing data also consumes more physical disk space.

SQL*NET (ORACLE) DATA SHARING
The above problems of sharing data in the TurboIMAGE environment can be addressed by utilizing Oracle's SQL*Net product. The data transferred in a SQL*Net environment will not be duplicated on the disk because the transfer is between the server disk and the client process. This product also provides new functionality of being able to update, delete, or insert data on either the HP or the VAX platform if both sets of data reside in a Oracle database. This functionality was not possible in a TurobIMAGE environment. Disk space is conserved because only one set of data exist and the operational problem of maintaining two sets of data and refreshing the remote data set do not exist in an Oracle SQL*Net environment. Software development becomes much more flexible and simplified because a single set of data can reside on either the VAX or the HP machine or any machine in the network.

NET BASE DATA SHARING
Netbase is a third party product that is currently being used to shadow and distribute our HP databases across our HP platforms to facilitate end user reporting, software development and online backups. This capability can be implemented in an Oracle SQL*Net environment in a way that will distribute the application across platforms. Distributing the application will conserve resources to perform this function. The server machine will extract the data while the client machine will receive and process the data. Oracle's solution will split up the amount of work to perform the data access between the client and server platform.

B. DATABASE RETRIEVAL METHOD

TURBOIMAGE RETRIEVAL
The typical user will not access the entire set of data but will access a subset of data. The method to retrieve a subset of data in traditional TurboIMAGE environment requires reading the entire data set. TurboIMAGE utilizes a hashed and linked list method of data retrieval. TurboIMAGE can retrieve records from the database very fast if the user knows the full key but can not retrieve on partial keys efficiently. A data access utilizing TurboIMAGE that retrieves a subset of data could translate into the machine reading one million records to create a report that may contain only one hundred records. The TurboIMAGE environment may require more machine resources and time to run a report on a subset of data. The expense of running reports under these conditions discourages

some reports from being run simply because the machine expense is too high to justify the resulting report. The report may also take so long that the data requested by the user becomes obsolete by the time the report completes.

ORACLE RETRIEVAL
Oracle provides a solution to this dilemma. Oracle utilizes a Btree retrieval and storage method that will directly retrieve a subset of data amazingly fast. Most users require data reports that are a subset of the entire set of data and Oracle performs partial key retrival very well. Each item of data in a Oracle keyed table within Oracle requires the same number of reads to reach a particular row in the database where as a TurboIMAGE request for data may require reading down a long chain to reach the one record the user is needing. A one hundred row report against a one million row table will only read one hundred rows in an Oracle database.

C. DATABASE NAVIGATION

TURBOIMAGE INTRINSICS
The traditional method of database navigation in TurboIMAGE is straight forware if the physical links of the database are followed. The physical links in TurboIMAGE must be defined at the database creation time. Typically all the possible reports which translates into data paths can not be anticipated at database creation time. A software developer can navigate outside of the physical links within TurboIMAGE, but this will result an increased burden on the developer. The software will be more machine expensive to run, and the software will be increasingly difficult to maintain.

ORACLE SQL ACCESS
In a relational database such a Oracle there are no physical links between sets of data. This approach allows the developer complete freedom in retrieving data in as complex or as trivial a method as required by the report. Also, Oracle provides access to its database utilizing the industry standard relational access method SQL. SQL simplifies the developers burden by grouping the data retrieval for the entire report into a single statement that is between 2 to 10 lines long. SQL will perform selection criteria and order the set of data, whereas TurboIMAGE will require the software developer to write additional software to perform these functions. SQL is used in 3rd generation languages such as FORTRAN as well as 4th generation languages as SQL*Forms and Speedware. This will allow the software developer to learn only one language (SQL) in all the tools that access a relational database. A common development language

will save retraining software developers when a new product is brought in. The TurboIMAGE method of data retrieval requires pages and pages of statements to extract the data and then additional pages to sort and select the data. The TurboIMAGE method of data retrieval quickly becomes very complex. This translates into an added burden on the developer, increased maintenance of software and delays software requests for the enduser.

D. DATABASE DESIGN AND DEVELOPMENT

TURBOIMAGE DATABASE CHANGES

A network database such as TurboIMAGE requires the developer to have the foresight to anticipate the data set relationships and the data items in the database. The current items and paths needed to describe the present business can be anticipated but as in any system, this will grow and evolve to keep pace with business changes. These business changes will command additional items to be added to a particular set, additional paths to be added to a set, or a complete redesign of a portion of the database. Making changes with TurboIMAGE requires a regeneration of the database or modifying the database using a third party package such as ADAGER or DBGENERAL. Changes also require the database not to be accessed during the database change which can leave the database unusable for hours. The production database is scheduled to be changed only on weekends, therefore, a software change that requires a database change made early in the week would have to wait to the next weekend to go in production.

ORACLE DATABASE CHANGES

One of the values of a relational database such as Oracle, is the ability to easily create the database and easily make changes to the database structure. A simple, one statement command can be used to alter or create a relational table of data. Relational database changes will occur instantaneously and can occur while the database is actively being accessed by users(the table that is changed must have a exclusive lock momentarily to alter the table structure). This ability is all included in the relational database package, where as TurboIMAGE needs a third party package like ADAGER or DBGENERAL to effectively make changes. The changes and modifications can also be made on line without impacting active users. This will relieve the operations staff from making changes at inconvenient times, such as weekends, and will indirectly reduce the overtime incurred by this task. Changes to the relational database are also instantaneous verses the hours required to rehash a TurboIMAGE database. Software changes that include a database change with the TurboIMAGE database could wait at least one week to schedule a time to alter the database.

RELATIONAL DATABASE DESIGN CONCEPTS

Relational databases like Oracle are designed following the established rules of relational database normalization utilizing tools such a entity relationship diagrams. The advantage of designing a relational database verses a network database, such as TurboIMAGE, is that the relationship between the tables of data do not have to be completely anticipated at database creation time. This relieves the designer of predicting the future relationships that business will demand of the database. Using the Oracle database will extend the life cycle of a database which reduces the time and volume of future database changes that the developer will have to implement.

One of the values of a relational database is that the knowledge and methods used to design and develop a relational database are standard and are based on a common set of rules. Relational databases should be the norm in the industry for the next 15 to 20 years. The industry is moving away from developing systems using network databases. Relational database concepts are currently being taught at universities and within the industry. New manufacturing systems are being developed utilizing relational technology. The entire software industry is looking toward opportunities to develop open systems and one way to do this is utilizing relational technology.

E. SOFTWARE DEVELOPMENT

APPLICATION DEVELOPMENT COMPARISON

Software development utilizing TurboIMAGE as the database requires the developer to perform many additional tasks that are not required in a relational database environment. A simple software change that requires an item change to the database can ultimately result in a tremendous amount of extra work for the developer. A program utilizing TurboIMAGE requires the length of the data record being read or altered be known to the program. If the program is doing a DBPUT or a DBUPDATE the programmer must remember to initialize the buffer. If the data record is changed to support a users request, this change will result in all programs that access that particular data set to be modified. One software change for a single program can balloon into possibly modifying n number of programs. This modification will result in a source code change and then a recompilation of the source code. Some or all of these programs that are affected may currently be checked out of the software control library which will delay the completion of the change. The developer can either wait for the software to be returned to the library or the developer can coordinate the change to be included with the prevailing software change. After the software changes to all the programs have been made, the database buffer length change and all the software changes must be installed on the production machine. The additional volume of software to install in a TurboIMAGE environment will make this a more difficult install.

A software developer may be faced with another problem related to a buffer size change. The developer may choose to create a new data set to support the database change rather than increase the buffer size. This may be forced on the developer for a number of reasons; the volume of software changes may be high, some of the software may not be able to be modified for a particular reason, or circumstances exist that prevent the change from being implemented in a timely fashion. This will breed many satellite datasets that are created solely because of the difficulty of implementing the change with TurboIMAGE. Extra disk space will be required to store these satellite data sets. Documentation, software development and software maintenance will become more complex because of the extra data sets.

The above problems can be prevented in an environment utilizing relational technology such as Oracle. Software developed utilizing Oracle products do not have to be recompiled if a column is added or altered in a table in the Oracle database unless the column is used in the software. The database change is easily implemented, instantaneous and does not require identifying and modifying all the software that access the data table. This will prevent the problem previously referred to, of creating extra data tables just to implement a software change. The coordination problem with implementation of software is removed because only the software that needs to be changed will be modified and implemented.

TurboIMAGE's method of retrieving a record of data creates an additional burden on the software developer. The developer must deal with the word offset from the beginning of the data buffer to access a particular item. If the offset is wrong, strange side effects will occur in the program. This situation will consume the time of the software developer chasing a software bug rather than implementing the software change intended. This can be facilitated with a set of include files for all the data sets in all the TurboIMAGE databases but the include files will require the additional burden of maintenance and software control. Software development in Oracle uses the columns name rather than the word offset. This relieves the developer of word offsets and record buffers. Oracles use of column names in a SQL statement rather than TurboIMAGEs use of word offset will make the program more self documenting and easier to maintain.

Oracle provides the developer with an active data dictionary that can simplify the software development process. TurboIMAGE applications do not use the item name to retrieve data from the database because all record data is actually transferred in one buffer. This causes problems because the TurboIMAGE database may refer to the item as "INTERNAL-P/N" while one program calls this same item "ASSEMBLY-IPN" and another program may call this same item "IPN" and so on. The point here is that there is not one single item name between the database and programs and additionally not one single item name from program to program. Software development utilizing Oracle products requires the column name to be the same as the name in the data dictionary within Oracle. The value is that a column name in all programs will be referred to by the same name in the data dictionary. Oracle's data dictionary also provides information about the column type and possible constraints for each column. A single name for a column is used within all programs and which allows the software developer to find what software is accessing a column in the database.

Oracle's data dictionary can define numeric items with a decimal point. The TurboIMAGE environment of software development uses implied decimals with integer types or uses real types and deals with rounding errors. There is no sure way to determine the implied decimal location except by looking in the software and it is possible that this software is in error. Software at times is installed with incorrect decimal location. Additional software must be written to handle the implied decimal. This problem does not exist in an Oracle development environment because of the Oracle handles numbers with decimals places.

Manufacturing applications store a multitude of date and time data to record particular events. TurboIMAGE does not provide the designer with a date data type. Databases developed with TurboIMAGE may store dates in a single word item using MPE's proprietary Julian date format. The software developer must use intrinsics to input, output or perform simple date arithmetic on the stored dates. There is no facility in QUERY to present the MPE Julian date in a Gregorian date format. This will restrict adhoc queries and program development that include date items. Oracle provides the developer the ability to simply output the date in an infinite number of formats. Oracle stores dates in the database as an actual date data type that includes the date and time. Date and time arithmetic can be easily performed on the date type column within Oracle. The input, output and formatting can be easily performed within the Oracle environment utilizing SQL*Plus.

LOCKING COMPARISON

TurboIMAGE provides database, set, and entry level locking that must be explicitly identified in the program being developed. Set level is currently used to lock our TurboIMAGE databases. Set level locking experiences high levels of contention because quite often more than one user will be accessing the same set of data at the same time which queues access. Oracle automatically provides a row level locking strategy that is managed at the database level and requires no software to be written by the developer to implement. Row level locking almost assures that there will never be a contention problem because two users rarely access the same row for update or delete at the same time. Locking a smaller object as moving from set level locking to entry level locking would reduce the locking contention within TurboIMAGE applications. This would be a tremendous effort because it would require all the software that accesses a particular data set to be completely changed from TurboIMAGE set level locking to entry level locking before any benefits would be realized by locking at the entry level. Entry level locking in TurboIMAGE will add additional lines of code and complexity to the software which will increase development and maintenance cost. This solution would add additional burden on the software developer and increase the code size which will indirectly increase maintenance cost.

TurboIMAGE escalates locks to a wait queue when more than one process locks the same object. Since set level locking is used this wait queue can become quite large and will directly impact user response time. Oracle locks at the row level therefore contention from locks is almost non existent. Where lock contention does occur in Oracle it is not escalated to a wait queue.

Oracle provides the user with a read consistent view of the database at all times. Oracle manages this feature automatically on the database level utilizing rollback segments. If user one is in the middle of a large transaction, user two will still be able to read a before transaction view of the data set. The advantage of this functionality is that readers of the database never need to block writers to get a consistent view of the database. TurboIMAGE recommends to lock if concurrent access occurs on a critical data set that is being reported. Locking the dataset will increase contention and slow response time. Oracle does not need to lock to assure a read consistent image.

TurboIMAGE will allow the developer to inadvertently dead lock the database. This situation is referred as a fatal embrace where the locks will continue to escalate until the system completely comes to a halt. The only solution to a dead lock is to completely shutdown the system. An abrupt shutdown will impact production by causing an unscheduled shutdown. Along with the shutdown, corruption can occur to the pointers within the TurboIMAGE database that may require pointers to be rehashed in the dataset. A dead lock is not possible in an Oracle database. Oracle will back out to this situation without impacting the system or users. This feature in Oracle is automatic and requires no software development.

ADDITIONAL ORACLE DEVELOPMENT FUNCTIONALITY

The size of third generation code size in a relational environment such as Oracle is much smaller than our TurboIMAGE environment. Smaller code size translates into quicker development and lower maintenance cost. Oracle application code is smaller in the critical input output areas of the application. Data in an Oracle application can be captured in a handful of statements that would require pages of code and complex looping structures to retrieve in a TurboIMAGE application.

Oracle provides the facility to create views of data with a stored query. The views can be used to pre-join several tables together to facilitate program development, adhoc reporting or enduser reporting. Views can also be used to slice and dice data vertically in a particular table into separate security objects. This is done at the database level so this would not require any additional software development. Views can also be used to present only a subset of columns to the user. This type of view can be further utilized to secure particular columns of data.

Oracle developers can use synonyms to redirect the process to another table or rename columns on a database level. This will provide data independence and location transparency. This will aid the developer in development and installation because the software can point to a development database or a production database anywhere on the platform or on any platform in the network without any software changes.

An active data dictionary that is provided within Oracle is superior to the independent data dictionary provided with 4th generation products that access TurboIMAGE such as Speedware and Powerhouse. An active data dictionary is an actual component of the database. Whenever a table is created, an entry is automatically entered into the Oracle data dictionary. In the independent data dictionary, the application developer must create the physical data set then upload the data set information from the TurboIMAGE root file. The information in the independent data dictionary must be further edited and manipulated to complete the data dictionary entry.

F. OPERATIONAL ADVANTAGES

DISK CONSUMPTION

Oracle consumes disk space very efficiently compared to TurboIMAGE. One particular set in TurboIMAGE consumes 11.5 megabytes while the same set of data consumes only 4.5 megabytes of data in an Oracle table. Oracle is able to conserve disk space by not storing blanks, nulls and by storing numbers in the minimum number of bytes needed to store (128 would only consume 3 bytes). One added benefit to this data compaction, is that an equal size disk IO on Oracle database will retrieve a considerably higher volume of actual data as compared to the same disk access against the same set of data on a TurboIMAGE database. In other words, more data will be transferred per disk access with data stored in an Oracle table.

DISK SPACE MANAGEMENT

TurboIMAGE space must be preallocated to each individual set within the database. Oracle preallocates space on a much larger object called the tablespace. Essentially, the entire database in an Oracle environment is allocated space. This relieves the operational staff from monitoring the hundreds of TurboIMAGE sets that have the potential of reaching capacity. TurboIMAGE manages space on the individual data set level, therefore, space can not be shared between datasets. Oracle's method of allocating space can conserve diskspace that is preallocated to the database because the pool of space can be shared between a larger number of objects.

IV. RECOMMENDATIONS

One of the points that we can learn from other shops that are implementing relational databases is DESIGN, DESIGN, DESIGN. A good relational database design will allow application development to progress smoothly, while a bad design can kill a project. Luckily, there is a method to design a relational database called normalization. There are also tools on the market that facilitate design and database normalization that should be considered. Database designs should be documented and presented to a review committee to obtain constructive feed back and hone the design. A database design must go beyond the existing application and consider all the connecting interfaces.

The database is the engine of an application. Applications developers are intimately familiar with the strengths, weaknesses and capabilities of the database for which they are developing software. Switching the engine from the network model TurboIMAGE to the relational model Oracle is a significant cultural change for the HP developers. Education is one of the major ways to instill a cultural change. This education will require a significant investment that must be considered.

Cultural changes usually take time to effectively take hold and implement. Lessons learned from other shops implementing a relational database, were to start small and gradually progress. A pilot project is recommended to begin implementation success. This pilot project is medium to small and it has the trait of exercising many of the features of the Oracle database. After successfully implementing a pilot project the development shop should be postured to move on to a larger project.

The network will become a very important part of our production system when we begin to install applications that do client server access over the network. The network will become as important as the production hardware that an application is running on. If the network goes down it will effect all the applications that are doing client server data access over the network.

One of the advantages of a relational database is that some of the functionality handled in the application program will now be handled at the database level. This saves the developer from repeating this functionality in all the programs that utilize this feature. This also requires that greater configuration and control be placed on the relational database environment. A database administrator would be responsible for much of the database level requirements that applications, systems and networks demand. There must be a person or persons who are responsible for the administration of the database. They must also have the knowledge to perform this function.

V. IMPLEMENTATION

This evaluation has identified several projects that could be used to implement Oracle. A new system has been ear-marked as a pilot project to kickoff Oracle implementation. This system fits the qualifications for a pilot project because it is a new system, it is medium to small in size, and it requires data to be shared between the VAX and HP platforms. This pilot will also lay the foundation to move the other systems to the Oracle database. The Information Systems group has been working closely with other groups to come up with an effective solution utilizing Oracle. Knowledge and techniques learned from the pilot project should be utilized to generate a set of standards that will be used to guide future development with Oracle.

Currently the manufacturing system contains several flat files. Flat files lack the support system that a database provides. Some of the flat files have a complex linked list data structure which adds an additional burden on the software developer. Oracle, or other more advanced databases, relieve the developer from the complexities of this data structure. They are also prone to corruption from time to time which may cause loss of data and down time to repair damage. Converting the manufacturing system flat files is a prime candidate for a follow up project after the pilot project. Many rewards could be reaped by converting these files to a relational structure. This will be a major project and should be approached with the required resources to design, develop and implement. All the outstanding software enhancement request that touch these files should be included in the design. The user community should be canvased to see if there are any unknown requests that need to be incorporated in the design. All this information should be incorporated into overall relational design.

There are a number of request for software changes on the HP and VAX development side that can not be done without data sharing between the VAX and HP platforms. These request should be addressed as the supply of Oracle trained developers become available. Having connectivity between platforms will open up more possibilities to combine data to answer questions.

Any new systems being considered that include a major change in the database design or that will create new database objects are prime candidates to be developed utilizing the Oracle database. Considering the added value that Oracle provides, it is prudent to develop these types of projects utilizing the Oracle database. Any new systems should also have a relational design and be implemented using the Oracle database.

VI. CONCLUSIONS

I was originally one of the skeptics of the advantages of Oracle on the HP platform. During the evaluation it became clear that this new technology, once implemented, could significantly help us develop and maintain software within our group. Daily problems that would surface with our current software applications were used to desk check a what if scenario in a relational environment. A relational environment could simplify, solve or eliminate many one of our current software development and maintenance issues.

TurboIMAGE is a technology that is nearing the end of its lifecycle. Many software vendors are moving to a relational database to develop there products. The market trend is certainly no reason to jump ship to relational database, but it is a barometer of the future trend. HP has hinted that their relational database will be the database of the 90's. HP's support and enhancement resources are being diverted from TurboIMAGE to newer technologies.

There is a cost of continuing to develop software with the traditional TurboIMAGE database. Software design, development and maintenance will be more expensive with the TurboIMAGE database. Operational costs are higher in the TurboIMAGE environment. There will be a hidden cost of the functionality that will never exist in the TurboIMAGE database that are available today with the relational database.

One of the advantages that can only be realized once Oracle is implemented is the connectivity between the VAX and HP platforms. This connectivity will allow users to get solutions from our Information systems that were not possible in the past or were difficult at best to implement. All hardware, software, and networking vendors are trying to achieve this goal. The Oracle database and networking products allow us to procure this capability today.

The intent of the evaluation was to stress the Oracle database to its limits and compare this with a similar TurboIMAGE application. The largest sets of data was used in the evaluation. One of the more complex data retrievals, the bill of material explosion, was demonstrated. The Oracle sales team was extremely supportive of our evaluation and surprisingly objective by openly discussing some of the weaker points of their products. This evaluation required the cooperation of all the major groups; HP development, VAX development, HP operations, VAX operations and Network systems folks. I would like to thank all the groups that participated and supported the evaluation. It could not have been done without your help. This cooperation is an indication that this new relational technology can be received and implemented.

Validating Your Database

What You Don't Know Does Hurt You!

presented by

Shelby Robert
Proactive Systems
Four Main Street
Los Altos, CA 94022
415-949-9100

With all the recent discussion of referential integrity in databases, it is surprising that nobody has raised the general question of what logical constraints should be imposed on production databases.

For example, it often arises from program bugs, faulty system design, system failures, program aborts or inadequate checking of entered data, that data in IMAGE databases is logically incorrect even though you may not be aware of the problem immediately. This can show up later in duplicate or surplus records, missing records, control totals not adding up, the presence of non-unique data-items, etc.

This article explains how it is possible to check that an IMAGE database is logically correct, using appropriate software tools.

WHAT IS LOGICAL DATA INTEGRITY?

Let's first establish a definition of "Logical Data Integrity" since the meaning of the term might not be immediately apparent. Indeed this is an area of integrity which has largely been ignored while users have trusted in the combination of IMAGE structure and good programming to keep data relationships in order.

What we are NOT concerned with in this discussion is structural data integrity -- that is correctness of links, paths and physical positions of data within files, all of which are the responsibility of IMAGE itself. The IMAGE database management system is generally very effective at maintaining the structural correctness of data, with only rare occurrences of problems such as "broken chains" appearing. So this discussion is looking beyond the correctness of data from the structural point of view, to ask: WHAT OTHER CONDITIONS SHOULD THE DATA SATISFY IN ORDER TO BE CONSIDERED FULLY CORRECT AND INTACT?

DATABASE DESIGN AND LOGICAL INTEGRITY

The additional constraints for logical integrity arise from the user environment and are conceptually determined by the database designer. Some of these conditions can be expressed in the database structure itself. For example a CUSTOMER's ACCOUNT-NUMBER must be unique, and this can be expressed and indeed enforced by making the customer set a master set with the account number as the search-item.

Another condition commonly enforced by design is represented by the following example. The TRANSACTIONS must not be entered into the database unless a CUSTOMER entry is present with the same ACCOUNT-NUMBER. This condition can be easily enforced by making the TRANSACTIONS set a detail set with a path to the CUSTOMER manual master set, again using the ACCOUNT-NUMBER as search-item. If this is done then any attempt to add a TRANSACTION which does not have its related CUSTOMER already in the database, will result in an error message from the IMAGE software.

You can see that some logical integrity requirements can be expressed, and thereby ensured, through the design of the database. There are other logical integrity requirements, however, which cannot be expressed in an IMAGE database design because there is no aspect of IMAGE which encompasses them, e.g. the checking of control totals.

Not only that, there can be situations where the performance overhead of including an extra path is not worthwhile. In other situations the requirement may not have been anticipated at the design stage, and it is too expensive to change the design at a later stage. So although purists may argue that normalization and design are the way to deal with logical integrity, in the real world there may be practical reasons why the user has a design which does not enforce the logical integrity requirement.

THE PROBLEMS WITH MAINTAINING INTEGRITY

So integrity requirements which are not enforced by the IMAGE database design must be dependent on the sustained accuracy of the application programs modifying the data. Such programs, particularly when they deal with the less used functions such as deleting records, may have hidden flaws which once in a while will leave a data relationship out of line. Application programs are enhanced or modified on occasions at which time there also is a greater vulnerability to error. A system failure or a program abort while in the throes of what should be considered a "logical transaction" can also allow errors to arise (a logical transaction is a series of database calls which must be completed as a group for the data to remain intact, whether or not this is formally made into a logical transaction by surrounding the calls with a DBBEGIN and a DBEND). And finally there is always the possibility of injudicious use of QUERY in deleting or adding entries without regard to logically related entries, which might be a cause of problems. Indeed to quote Murphy's Law -- *If something can go wrong then one way or another it probably will.*

There are various consequences of data getting out of line in these ways. At first sight the consequence of leaving a delivery address on file when a customer has been deleted may seem trivial, but what happens if the account number is re-used? Probably an abort from an IMAGE "Duplicate-Search-Item-Value" when an attempt is made to DBPUT the new delivery address. Or imagine that an obsolete product is deleted from the database without reference to an outstanding order-line for that product. At some stage, in some run or other, the failure to reference the product code is likely to cause an abort.

There is little point in listing further hypothetical problems, but two characteristics of this sort of problem are worthy of note. Firstly the problem is quite likely to bring down a live batch run, and secondly the problem will be unusual, if not unique in its occurrence. This may result in a problem which is difficult to identify, certainly difficult for the end-user to identify, in a scenario where the flow of work is waiting on its being fixed. So while I do not want to appear alarmist, the reality is that many, if not most, databases do have data relational requirements which are not enforced by the database design. These requirements can be violated in a variety of ways, and the mismatches that result can potentially give rise to occasional bizarre or obscure problems.

This article is about the possible ways of checking or verifying the logical relationships within the data. To that end it will also be necessary to identify as many as possible of the different types of data relationship which are commonly used in databases.

CHECKING LOGICAL DATA RELATIONSHIPS

There are basically three approaches to verifying logical data relationships. Firstly it is possible in some instances to use QUERY (or other reporting systems) to identify missing data relations or to perform validation of a data-item value. Secondly custom-built pieces of software can be written which are tailor-made, effectively as part of the application, to check the individual requirements of logical integrity. This approach is obviously demanding of developers' resources. Thirdly it is possible to use generic software developed specifically to check and validate logical integrity. Our company has always looked toward developing generalized solutions to problems and our own module within the FLEXIBASE software is the creative source behind this discussion.

A generic validation program needs to be able to identify the common logical requirements, hereafter called rules. It also needs to allow the user to easily introduce into those rules the specific conditions that need to be checked.

This generic software should be able to accept as many rule specifications as are necessary and then do all the processing necessary for checking those rules in one single pass of the database (preferably by a simple serial read of the data sets involved).

The resulting output should list only the exceptions that have been found so that the user can quickly locate and deal with the problems. Software of this type should be capable of being run on a regular basis, so as to ensure that the data maintains its logical consistency. Such software can improve the reliability of processing, and save time which would otherwise be spent on resolving discrepancies, because it will routinely identify potential problems.

Access to such software actually opens up other possibilities beyond the simple checking of integrity. It can additionally serve in system testing as a debugging aid on user programs which are required to maintain such logical relationships. It can also have uses in identifying unusual or conditional situations requiring management attention. Finally the availability of this kind of software may allow users to remove certain paths or make other structural simplifications which would lead to improved performance, since the constraints that are removed can be checked by periodic runs of the software.

THE RULES OF INTEGRITY

There are six rules that you commonly want to check, which are as follows:

RULE 1: RELATIONAL CORRESPONDENCE

Relational correspondence, in non-jargon terms, simply means that if a data-item value is to exist in one place (i.e. set or table) then it must also already exist in another specified place. Often this is automatically enforced by the IMAGE database design such as when TRANSACTIONS cannot be added unless the CUSTOMER chain-head is present. For the purpose of our discussion the most straightforward example requiring checking is the existence of a DELIVERY-ADDRESS for each CUSTOMER; and vice versa where there must be a CUSTOMER in existence for each occurrence of a DELIVERY-ADDRESS. It might not be unusual for this rule to be applied in both directions. Alternatively if not all CUSTOMERS have DELIVERY-ADDRESSes then the rule would only be applied in one direction (to check that every DELIVERY-ADDRESS is for a CUSTOMER who does exist). The general form of the rule reads:

> *For each entry in set(A), using the value of item(X) as search-item, there must be a corresponding entry in master-set(B).*

An example of the use of this rule might be as stated above. If the user substitutes for (A), (X) and (B), the names CUSTOMERS, ACCOUNT-NO and DELIVERY-ADDRESS, then the rule reads:

For each entry in set(CUSTOMERS), using the value of item(ACCOUNT-NO) as search-item, there must be a corresponding entry in master-set(DELIVERY-ADDRESS).

So applying this rule would check that for every customer, there is in existence a delivery-address, and would report any customers for which there is not a delivery-address.

The rule could also usefully check for relational correspondence in a different database. So if in another database called STATS there should be an entry for each CUSTOMER in its CUSTOMER-ANAL set then the rule should read:

For each entry in set(CUSTOMERS), using the value of item(ACCOUNT-NO) as search-item, there must be a corresponding entry in master set (STATS:CUSTOMER-ANAL).

Database designers sometimes create the situation where the related key value is actually a partial or concatenated key i.e. a piece or pieces of a data-item, or more than one data-item joined together.

For example if the 5th and 6th characters of the account-no are the Market-Code, then a check could be made that an entry for each Market referenced by a CUSTOMER does exist:

For each entry in set(CUSTOMERS), using the value of item(ACCOUNT-NO[5,2]) as search-item, there must be a corresponding entry in master-set (STATS:MARKETS).

A more complex example might read:

For each entry in set(TRANSACTIONS), using the value of item(ACCOUNT-NO[3,4]+MARKET-CODE+PRODUCT-CODE[1,2]) as search-item, there must be a corresponding entry in master-set(SALES-ANALYSIS).

The checking of this rule can be implemented, albeit inefficiently, by using a special parameter of the MULTIFIND command within QUERY after a JOIN which uses the parameter. When a MULTIFIND command is executed and corresponding entries are missing, then the value of the items in the missing entries are given the value $MISSING within QUERY. Thus the following QUERY syntax would implement a search for missing DELIVERY-ADDRESSes as per our example above assuming DELIVER-TO is one of the item-names in the DELIVERY-ADDRESS set:

```
>JOIN CUSTOMERS.ACCOUNT-NO @ TO DELIVERY-
ADDRESS.ACCOUNT-NO

>MULTIFIND ALL CUSTOMERS.ACCOUNT-NO

using serial read

5486 compound entries qualified

>SUBSET DELIVER-TO=$MISSING

4 compound entries qualified

>R ALL .......
```

Up to now we have used this rule for discovering MISSING entries i.e. the software would report exceptions to the rule where the related entry is not found. The rule can also be useful with its logic reversed e.g. if the CUSTOMER exists in the DIRECT-SALES set then it must NOT also exist in the AGENT-SALES set. In this form the rule might also have management reporting uses e.g. "Which of our PRODUCTS are also in our RAW-MATERIALS set?"

RULE 2: VALUE OF DETAIL-CHAIN TOTAL MATCHES CHAIN-HEAD

For each entry in master-set(A), check that the total of items(X+Y..) is equal to the total of items(P+Q..) from all the entries of its chain into detail-set(B).

An obvious example for this relationship is where the total of the CREDIT and DEBIT amounts in each of the TRANSACTION detail entries in a customer's chain should add up to the BALANCE as held in the CUSTOMER master entry itself. So the substituted rule might read:

For each entry in master-set(CUSTOMERS), check that the total of items(BALANCE_BALANCE-FORWARD) is equal to the total of items(DEBIT+CREDIT) from all the entries of its chain into detail-set(TRANSACTIONS).

I have used the underscore character here representing the minus sign so as to avoid confusion with the hyphens in item-names. The rule and its uses are fairly self-evident, but note the importance of being able to sum more than one field in each entry so as to be able to cope with separate debit and credit items, balance-forward as well as current-balance etc. This generic function might well find favor with auditors as well as bookkeepers.

While it is theoretically possible to generate a report in QUERY which would provide both chain-head and chain totals, it is not possible to print only the exceptions, so much printing and manual review of the printed matter would be necessary.

RULE 3: DETAIL-CHAIN LENGTHS

For each entry in master set(A), check that the length of its detail chain into set(B) is (=><) length (n).

A common example might be where there should be a one-to-one relationship between an automatic and detail set. This is often set up by designers when it is more convenient to put the data into a detail set, but direct keyed access is still required. Although IMAGE would ensure that an automatic entry must always be present for each detail entry, it is still possible for there to be more than one detail entry for each automatic master entry. So the rule could be used to check that every chain-length is equal to 1.

Another common situation is where there should always be at least one detail entry chained to each master entry, so the rule could be used to check that all chain-lengths are greater than 0.

In some circumstances there should be a specific number of entries on each chain -- one for each class of product, or maybe there should never be more than a specified number of product classes. As a management information function this rule could be used to list exceptional conditions such as more than 20 outstanding INVOICES or less than 2 TRANSACTIONS in a year, or more than one DELIVERY-ADDRESS, or to identify inactive accounts where there are no current TRANSACTIONS, or dead ORDERS where there are no outstanding ORDER- LINES.

Note that allowing the user to specify any of the three conditions = < > and also to specify the actual chain length for comparison makes this rule very flexible and comprehensive while still being simple in construction.

RULE 4: UNIQUE DATA-ITEM VALUES

For each entry in set(A), check that the value of data-item(X) is unique.

IMAGE enforces the uniqueness of search-item values in a master set, but it might be required to maintain uniqueness of the value of some other item in a detail or master set. For example if each INSURANCE-CLAIM is a detail entry chained to the CUSTOMER master, we could check the uniqueness of CLAIM-NO by:

For each entry in set(INSURANCE-CLAIMS), check that the value of data-item(CLAIM-NO) is unique.

This rule can also identify duplicate data-item values. Suppose that it is suspected later that one or more batches of invoices have been posted twice. The duplicate entries could be identified by using the rule:

For each entry in set(TRANSACTIONS), check that the value of data- item(REFERENCE) is unique.

The rule could also check the viability of potential search-items before a structural change turning a detail-set into master-set. Since the data to be checked could be a concatenation of data-items or pieces of several data-items, as discussed under rule 1, complex checks of uniqueness or duplication can be made.

RULE 5: PARALLEL DETAIL CHAINS

For each entry in master set(A), check that the length of its detail chain into set(B) is (=><) the length of its detail chain into set(C).

For example perhaps each entry in the master-set CUSTOMER should be linked to the same number of TRANSACTION entries as SALES-ANALYSIS entries. This rule could check for equal lengths.

It sometimes happens that where an entry size becomes very large, that the user will decide to split it into, or define it initially as two parallel detail sets, and this then implies that the chain lengths into each of the sets should always be the same. This rule can check that situation. Alternatively this rule could identify where there might be less INVOICES than DELIVERIES etc.

Again the facility to use one of the three conditions = < > makes the rule very flexible.

Note that again QUERY can be of some value in this check. Firstly using the FORM command will give entry counts which can be useful in the case of equal chain-lengths, although it will not show-up compensating errors. The definition of the set may also lend itself to using the MULTIFIND $MISSING construct as discussed under rule 1.

RULE 6: SUB-ITEM TOTALS

For each entry in set(A), check that the total of all subitems in data- item(X) is equal to the value of data-item(Y).

Because of the arithmetic limitations in some report-writers, for ease of reporting, users sometimes hold an annual total in one data-item as well as the individual month values in a separate sub-item array. This rule will routinely identify where the two are not equal:

> *For each entry in set(SALES-ANALYSIS), check that the total of all sub-items in data-item(MONTHS-SALES) is equal to the value of data-item(TOTAL-SALES).*

CONCLUSION

It is clear that for various reasons the database design is not sufficient to enforce all the conditions which data must satisfy in order to be considered correct and intact. There are many types of events which can cause data to deviate from these conditions, and the consequences are unreliable processing. Six rules were identified and discussed -- while not exhaustive -- probably cover the great majority of integrity requirements. Mostly these concern the relationships between entries in different sets. The application of generic software or QUERY to the checking of data integrity was illustrated using these rules as a framework. You can improve the reliability of your application systems by running a periodic check and taking corrective action before faults propagate through your database.

Paper No. 3157

SQL for the Curious

by

Dirk Huizenga

Dynamic Information Systems Corporation
5733 Central Avenue
Boulder, CO 80301
303-444-4000

Why SQL?

Structured Query Language or "SQL" has been around since about 1976 when IBM began work on a revised version of "Structured English Query Language" (SEQUEL). This development was the result of work by E.F. Codd the author of *A Relational Model of Data for Large Shared Data Banks.* Several prototype relational languages came out of that research. One of these languages was SQL.

In the late 1970s several companies, including IBM, started development of relational products. The first to market was a product called ORACLE. In 1981, IBM announced a SQL product called SQL/DS for the DOS/VSE environment and later (1983) DB2 for the MVS environment.

Soon numerous other vendors came out with SQL-based products including SYBASE (from Sybase Inc., 1986) and INGRES (Relational Technology Inc., 1981, 1985). By the mid-eighties, versions of SQL ran in some form on micros to mainframes and was becoming the defacto standard for accessing relational databases. In 1986, the American National Standards Institute (ANSI) ratified the X3H2 proposal for a standard relational language making SQL the *official* relational language.

Why do we need SQL? Several reasons in favor of a standard query language, such as SQL, are:

- Application portability
- Extended application life
- Multiple platform communication
- Reduced training costs

These reasons could be applied to any query language, existing or in the future. In its favor, SQL exists as a standard, as numerous products, and is not dominated by any one company. This is how COBOL has survived for so many years despite numerous obituaries.

This does not mean that SQL is perfect. Chris Date included a critical analysis of the SQL language in his book, *A Guide to the SQL Standard*, as well as his own comments critical of the standard. Having a standard means half the battle has been fought. What happens now is up to you.

The Big Picture of SQL

The function of SQL is to support the creation, configuration, and manipulation of a relational database. SQL is actually three languages in one: a data description language (DDL), a data manipulation language (DML), and a data control language (DCL). SQL tries to minimize the effort a programmer must expend to retrieve data. It is a nonprocedural language which is used to indicate **what** is desired rather than **how** it might be achieved.

Common Ground

SQL and relational databases do not define the physical file structure that underlies the language. The *Encyclopedia of Computer Science and Engineering* describes a relational database as "a network database in which the relationships are handled in a very specific way."

SQL combines the functions of Dbschema, Dbutil, and Query all into one program. It has commands that define the tables (sets) and columns (items). It also has commands that can retrieve rows (records) from a relational database. These commands can produce basic report, similar to the List command in Query. To create more sophisticated reports you must use the embedded version of SQL. There are also commands to control access to the data by various means.

In the Hewlett-Packard's ALLBASE/XL version of SQL, the interactive version is referred to as ISQL and the embedded version is simply SQL. The examples in this paper will be using the interactive ISQL. Although SQL will accept commands in upper or lower case the following conventions will be used here to avoid confusion:

- All SQL command keywords will be shown in uppercase.

- Names of tables, columns and user-defined terms will be shown in lowercase.

- All SQL commands will be terminated by a semicolon (;).

Data Definition

To define a database in ISQL you specify several DDL statements that are grouped by into one or more schemata. Each schema is made up of two kinds of tables: base tables (or just tables) and viewed tables (views). A base table is a "real" table or physical file. By contrast a view is a virtual table that does not exist in physical storage, but is made up of one or more base tables. The following three SQL statements creates a ALLBASE database that will hold stock market information. To execute these commands interactively, you should run ISQL.PUB.SYS.

```
START DBE 'stocks' NEW
      DBEFILE0 DBEFILE stocksdbe0
      WITH PAGES=150,
      NAME='stocksf0';
```

The START command is not part of the SQL standard, but is required by HP SQL to define the physical environment including space for tables and indexes. The PAGES parameter is where physical space is specified. Additional DBEFiles can be added to the database as more space is needed. The *ALLBASE/XL HP SQL Database Administration Guide* contains more information on setting up the physical environment. Now on with the example.

```
CREATE TABLE company
      (
      symbol            CHAR(5) NOT NULL,
      company_name      CHAR(40),
      exchange          CHAR(10),
      date_entered      DATE NOT NULL,
      date_updated      DATETIME NOT NULL,
      address           VARCHAR(120),
      date_lastprice    DATE,
      lastprice         DECIMAL(9,3)
      );
```

```
CREATE TABLE prices
        (
        symbol              CHAR(5) NOT NULL,
        period              CHAR(1) NOT NULL,
        date_record         DATE NOT NULL,
        date_entered        DATE NOT NULL,
        date_updated        DATETIME NOT NULL,
        hiprice             DECIMAL(9,3),
        loprice             DECIMAL(9,3),
        closeprice          DECIMAL(9,3) NOT NULL,
        pe                  SMALLINT,
        dividend            DECIMAL(9,5),
        netchange           DECIMAL(9,3),
        seqno               SMALLINT
        );
```

HP SQL does not support the definition of a UNIQUE column in the CREATE TABLE command as does the SQL standard. To prevent having duplicate values in a table an index table must be created using:

```
CREATE UNIQUE INDEX symbolindex
        ON company (symbol);

CREATE UNIQUE CLUSTERING INDEX priceindex
        ON prices (symbol, period, date_record);
```

The index for the Company table will prevent duplicate stock symbols and speed retrieval. The index for the Prices table clusters records near other records with a similar key values, in addition to preventing duplicate key values. Clustering physical arranges records on physical pages to speed access to records when accessing records by that index. Indexes and clustering enhance retrieval performance, to the detriment of update performance. Indexes can be easily removed using the DROP command which is not a part of the SQL standard language.

Views

Views are tables that do not exist, but are made up of columns from base tables. A view can be made up of a subset of the rows in a single table or multiple tables. These tables can be accessed like any other table, but they do have some restrictions. A complete list of these restrictions can be found in the *ALLBASE/XL HP SQL Reference Manual*. The following command creates a view that is a subset of the original table.

```
CREATE VIEW weeklystats
        (
        symbol,
        date_record,
        volume,
        hiprice,
        loprice,
        closeprice,
        netchange
        ) AS SELECT
        symbol,
        date_record,
        volume,
        hiprice,
        loprice,
        closeprice,
        netchange
        FROM prices
        WHERE period = 'w';
```

Data Manipulation

Once you have defined a database, you can manipulate one or more rows using SQL's DML. It has four main verbs: insert, select, update, and delete. The following examples are ways of entering data using ISQL commands.

```
INSERT INTO company VALUES
        (
        'stk',
        'Storage Technology, Inc.',
        'nyse',
        CURRENT_DATE,
        CURRENT_DATETIME,
        'Announced earnings affected by delay of Iceberg',
        'Louisville, CO',
        '1992-05-29',
        35.125
        );
```

The interactive INSERT can only add one row at a time. ISQL's LOAD command can be used to add multiple rows to a table from a file or an application program can use HP SQL's bulk operation facility. The CURRENT_DATE and CURRENT_DATETIME are HP SQL functions that provide current date and time values. The next example of the INSERT command can be used to enter only certain columns as long as the omitted columns are not defined with the NOT NULL phrase.

```
INSERT INTO prices
        (
        symbol,
        period,
        date_record,
        date_entered,
        date_updated,
        volume,
        hiprice,
        loprice,
        closeprice,
        pe,
        dividend,
        netchange,
        seqno
        )
        VALUES
        (
        'stk',
        'w',
        '1992-06-05',
        CURRENT_DATE,
        CURRENT_DATETIME,
        16420,
        38.000,
        35.125,
        37.000,
        16,
        0,
        1.25,
        NULL
        );
```

The SELECT statement is the core of the SQL language. It lets you choose rows from the database. The simplest form of the command is

```
SELECT * FROM company;
```

The asterisk (*) specifies all the columns in the Company table without having to list them. If only a subset of the columns are needed, then the following statement can be used.

```
SELECT company_name, date_lastprice, lastprice
        FROM company;
```

So far these commands would select all the rows from the tables. To retrieve only a subset of the rows use the WHERE clause.

```
SELECT date_record, closeprice
        FROM price
        WHERE symbol = 'stk';
```

The Where clause can be used for complex search criteria that involve SQL functions, logical operators, ranges, comparison, and other SQL clauses. These queries can also span several tables or contain mathematical functions like AVG, MIN, MAX, SUM, and COUNT. For more information about the many ways of using the SELECT statement consult the *ALLBASE/XL HP SQL Reference Manual.*

You can modify values in selected rows using the UPDATE command. This statement uses the WHERE clause to specify the rows you want to change. To update the last price information for Storage Technology, enter the statement

```
UPDATE company
        SET date_lastprice = '1992-06-05',
        lastprice = 37.000,
        date_updated = CURRENT_DATETIME
        WHERE symbol = 'stk';
```

The DELETE command can use the WHERE clause to select rows for deletion. To delete all the stock price information before January 1, 1991 use:

```
DELETE FROM prices
        WHERE date_record < '1990-01-01';
```

To delete all the rows in the Prices table, just leave off the WHERE clause.

What can you do if you just deleted all the rows in a table unintentionally? Use the ROLLBACK WORK command. The ROLLBACK statement provides a way of terminating a series of commands called a SQL transaction. This prevents any updates from being done to the database. To end a SQL transaction normally, use the COMMIT WORK command to cause all database updates to be made to the physical files. Until a COMMIT WORK takes place any changes to the database, made by SQL commands, are not visible to other users. To improve concurrent performance, keep transactions short by using COMMIT WORK (or ROLLBACK WORK) frequently.

Using SQL with Third Generation Languages

Up till now the we have been discussing the interactive version of HP's SQL. To use the power of a SELECT statement in a COBOL program use the embedded version of SQL. To retrieve information from the Company table the following statement would be embedded in the PROCEDURE DIVISION of a COBOL program.

```
EXEC SQL
     SELECT COMPANY_NAME, LASTPRICE,
     DATE_LASTPRICE
     INTO             :NAME,
                      :PRICE,
                      :DATE
     FROM COMPANY
     WHERE SYMBOL = :TICKERSYMBOL
     END-EXEC.
```

All embedded SQL statements begin with "EXEC SQL" and end with "END-EXEC." The variables defined in the program for use in the SQL statement are prefixed with a colon (:). These host variables must appear in a special declaration area in the DATA DIVISION. Most SQL commands can be embedded into a program, including DDL commands to create tables, indexes, and views. DML commands can be used for various types of operations:

- Simple data manipulation: operations on a single or limited number of rows.

- Sequential table processing: operations that use a cursor to operate on a row at a time within a set of rows. A cursor is a SQL object, essentially a kind of pointer, that can be used to run through a collection of rows.

- Bulk operations: operations that affect multiple rows with a single DML statement.

- Dynamic operations: operations defined at run time.

For more information about using SQL from within a third generation language refer to the appropriate SQL language guide from Hewlett-Packard.

Conclusion

This quick tour of the SQL language is intended to satisfy you curiosity about what may be the biggest thing to come along since COBOL. You should be able to create a simple database and add, delete, and report data from it. Once you are familiar with SQL's terminology and concepts, it is easier to compare it to traditional database functions.

Standardizing on SQL allows an organization to use one language to access databases on the full range of computing platforms, from microcomputers to mainframes. It is very important to develop your knowledge of SQL as more vendors support this language standard.

Bibliography

Chou, George T., *Using SQL,* Que Corporation, 1990.

Date, C. J., *A Guide to the SQL Standard,* Addison-Wesly Publishing Co., 1988.

Hewlett -Packard Company, *ALLBASE/XL SQL Reference Manual,* Hewlett-Packard Co., 1989.

Hewlett -Packard Company, *ALLBASE/XL ISQL Reference Manual,* Hewlett-Packard Co., 1988.

Hewlett -Packard Company, *ALLBASE/XL SQL Database Administration Guide,* Hewlett-Packard Co., 1989.

Hewlett -Packard Company, *ALLBASE/XL SQL COBOL Application Programming Guide,* Hewlett-Packard Co., 1989.

Ralston, Anthony and E. D. Reilly, Jr., Ed., *Encyclopedia of Computer Science and Engineering,* 2nd Edition, Van Nostrand Reinhold Co., 1983.

TurboIMAGE/iX's Standard Interface to Third-Party Indexing

Presented by Donna Bloechl
Written by Eric Savage
Dynamic Information Systems Corporation
5733 Central Avenue
Boulder, Colorado 80301
(303) 444-4000

At the 1990 Interex conference in Boston, Hewlett-Packard announced it would provide a direct interface within TurboIMAGE/iX to third-party indexing packages. This would provide the capabilities of generic key access, sorted sequential retrieval, keyword searches, and multiple key searches from within TurboIMAGE/iX. Over the past year, the basic features provided by third-party indexing packages were standardized, and TurboIMAGE/iX was enhanced to provide direct access to these features. TurboIMAGE/iX now provides a better integration of these products, and allows applications to be developed without advance knowledge of which product is installed.

This paper describes the specific enhancements to TurboIMAGE/iX. It should be noted, however, that this interface is still in development. While the enhancements described here reflect the design as of this writing, Hewlett-Packard, and the third-party indexing vendors reserve the right to alter it as appropriate.

Overview of Interface

Hewlett-Packard will provide a generic interface for third-party indexing products. This is being done so that new methods of access - generic key access, sorted sequential retrievals, keyword searches, and multiple key searches may be provided. These capabilities are described below:

- ❑ Generic key access is the ability to use wildcards in the arguments of keyed searches. For example, the argument 'INFO@' qualifies all records beginning with 'INFO'. Users have not been able to achieve these searches in TurboIMAGE/iX previously, since TurboIMAGE/iX requires full key values in all keyed retrievals.

- ❑ Sorted sequential retrieval is the ability to return qualified records in sorted order along a key. For example, users of an accounting database may want to retrieve records from a set in several sort orders. One sort order could be date and then account code; a second sort order could be account code and then date. Users have only been able to accomplish limited sorted retrieval in TurboIMAGE/iX using sorted chains.

- ❑ Keyword search is the ability to qualify records based on individual words or values contained in fields, rather than full key values. For example, keyword search would allow a user to specify the word "Packard" and qualify the record "Hewlett Packard Company". Users have not had this capability in TurboIMAGE/iX.
- ❑ Multiple key retrieval is the ability to use more than one key in the same search. For example, when searching for all people named Smith in the state of California, multiple key search allows both the Name key and the State key to be used. TurboIMAGE/iX has allowed users to use one key at a time.

This interface is being provided in two fundamental forms. First, the TurboIMAGE/iX intrinsics have been expanded to provide direct access to third-party indexing. The searches mentioned above may be performed through new modes of DBFIND and DBGET. As well, all update and recovery operations to the third-party indexes are automatically invoked whenever TurboIMAGE/iX updates or recovers its own datasets. Second, the TurboIMAGE/iX utilities have been expanded so that they perform their functions against the third-party indexes when appropriate. For example, when a database is purged using DBUTIL, the third-party indexes are purged as well. When a database is copied using DBCHANGE PLUS, the third-party indexes will be copied as well. DBUTIL may also be used to enable or disable third-party indexing for each database.

To accomplish this interface, the third-party vendors will move their products into PUB.SYS, and their internal routines will be called directly by the System XL. The physical routines will not be located in the System XL, but instead in a secondary XL in PUB.SYS. This allows integration into the operating system without requiring operating system updates to install new versions of the third-party packages.

A Standard Interface to Third-party Indexing

The services and approaches offered by third-party indexing vendors differ. Hewlett-Packard has established a standard interface for third-party indexing which all vendors agree to provide. This standard interface establishes a common approach to the basic capabilities of third-party indexing, using common modes, arguments, status conditions and programming constructs. This standard[1] interface allows end-users and software vendors to develop applications that take advantage of

[1]The use of the word 'standard' does not indicate a standard beyond the scope of TurboIMAGE/iX. It is not comparable to the SQL or ANSI standards. Instead, standrd refers to the minmum set of features in TurboIMAGE/iX that all third party vendors agree to provide.

third-party indexing, even if they do not have advance knowledge of which third-party indexing package is installed on a database. This standard interface also provides the means to determine at runtime whether third-party indexing is installed, thereby allowing applications to use their indexes if they are available, and to use regular modes of TurboIMAGE/iX if they are not.

The third-party indexing products may provide features beyond those that are standardized. For this purpose, specific ranges of modes, status conditions, and argument tokens within the TurboIMAGE/iX intrinsics has been set aside. Each third-party vendor may use these at their own discretion, and for their individual purposes. Additionally, the third-party vendors may include additional intrinsics that provide features not directly translated into the TurboIMAGE/iX intrinsics.

Installing the Standard Interface

Installing the standard interface to third-party indexes will require three steps:

- ❑ Users must receive an operating system update from Hewlett-Packard containing the new interface. At the time of this writing, Hewlett-Packard is announcing that this interface will be available in the summer of 1992. Users must also receive a version of their third-party indexing package which includes this interface.
- ❑ Users must install third-party indexing on the database.
- ❑ Users must use DBUTIL to enable this interface for each database enhanced with third-party indexing. NOTE: Only one third-party indexing package may take advantage of this interface on a database at a time.

Updating the Database Using the Standard Interface

Once a database has been enabled for indexing using this standard interface, the indexes will be automatically updated every time a TurboIMAGE/iX intrinsic is called. No changes to programs are required. Previously, programs using third-party indexing needed to be run through external libraries (SL's or XL's) in order to update the database. This will no longer be necessary since TurboIMAGE/iX invokes the third-party indexing package as appropriate.

Enhancing Existing Retrievals

The DBFIND and DBGET intrinsics have been expanded with new modes, arguments, status conditions and programming constructs. These are needed to provide expanded capabilities of third-party indexing. In one situation, however, existing retrievals can be enhanced without programming changes.

One enhancement in the interface allows wildcards to be used in the arguments of a DBFIND call on a TurboIMAGE/iX detail search item. This means that users who have existing applications which perform keyed retrievals on detail sets can enter wildcards without any changes to the program.

Performing Third-party Indexed Retrievals

Aside from the situation just described, third-party indexed retrievals require using new modes, arguments, and programming constructs. There are two principle retrieval constructs, both of which use DBFIND and DBGET. In both situations, DBFIND is used to qualify the search, and DBGET is used to retrieve the actual records.

In both methods, DBFIND qualifies a 'working set' of records. DBGET is then used to move forward and backward along this qualified set, much the same way that DBGET is used to navigate TurboIMAGE/iX detail chains. These working sets are not fixed, though, as they are in TurboIMAGE/iX detail chains. The working sets are produced dynamically based on the criteria entered by the user. In generic key retrievals, the criteria 'INFO@' produces a working set of all records beginning with 'INFO'. The criteria '=910101' produces the working set of all records with a date greater than 1/1/91, in sorted order. In single and multiple keyword retrievals, the successive qualifications produce a subset of records that meet all criteria, and this becomes the working set.

For generic key access, DBFIND mode 1 is used, optionally with wildcard criteria. However, DBFIND is no longer restricted to TurboIMAGE/iX detail search items; this interface will allow DBFIND to be called on any indexed field in any type of dataset. For example, if a third-party key was installed on a field in a master set, then DBFIND would be called on that field in the master set. Additionally, the argument may contain wildcards and relational operators. After the DBFIND call, DBGET modes 5 and 6 may be used to retrieve records from this working set.

For single or multiple keyword retrieval, DBFIND mode 12 is used for each step in the retrieval. If multiple keys are used in a search, DBFIND mode 12 is called in succession for each key. After the records have been fully qualified, DBGET modes 25 and 26 may be used to retrieve records from this working set.

Figures 1, 2 and 3 show the DBFIND and DBGET intrinsics in more depth, as well as the new programming constructs which have just been discussed.

Developing Third-party Applications

As mentioned earlier, this standard interface allows end-users and software vendors to develop applications which take advantage of third-party indexing, even if they do not have advance knowledge of which third-party indexing package is installed on a database. This standard interface also provides the means to determine at run time whether third-party indexing is installed, thereby allowing applications to use their indexes if they are available, and to use regular modes of TurboIMAGE/iX if they are not.

To facilitate the development of these applications, new modes of DBINFO are provided which determine if a third-party indexing product is installed, and describe the installation of keys. By using these new modes, developers can fully customize their programs to use third-party indexing.

Summary

The TurboIMAGE/iX interface to third-party indexing is designed to provide end-users and software vendors a standard method to include the single-most requested IMAGE enhancement: generic keyword and multiple keyword access to IMAGE data. Users of this interface can expect to dramatically speed their keyed access retrievals, giving them far more flexibility in accessing the data they need for their daily operations and decision-making.

Any questions or clarifications on the interface should be directed to the HP Development Labs. Questions concerning either of the third-party interface products should be directed to the third-party vendor.

Figure 1 - New DBFIND Intrinsic

(Note: This is not a complete list of modes and argument tokens. Please consult the Omnidex manual for a complete list.)

DBFIND(base, set, mode, status, item, argument)

Mode	Argument	Description
1	Full key value	Standard search on TurboIMAGE/iX detail search items.
	Wildcard value, using: @ any numnber of letters or digits ? Any single letter or digit # Any single digit > 'Greater than' >= 'Greater than or equal to' <= 'Less than' <= 'Less than or equal to'	Generic key access on character type TurboIMAGE/iX detail search items, and on IMSAM keys in any dataset.
10	Full key value	Standard search on TurboIMAGE/iX detail search items. Guaranteed to bypass Omnidex and IMSAM.
11	Two adjacent binary values	Binary range retrieval. Start of range and end of range are expressed as binary values in argument.
12	String of keywords, using wildcards, Boolean operators, relational operators, and parentheses. The following are supported: @ any numnber of letters or digits ? Any single letter or digit # Any single digit > 'Greater than' >= 'Greater than or equal to' <= 'Less than' <= 'Less than or equal to' TO Inclusive range NOT'And not' AND'And' OR'Or'	Keyword retrieval, qualifying all records containing specified keywords anywhere in key. If the argument begins with a Boolean operator, the results of this DBFIND will be applied against the working set generated by the preceding DBFIND.
13		Undoes the last retrieval, and restores back to the most recently qualified list.
21	Same as mode 1	Same as mode 1, but does not return qualifying count.

Figure 1 - New DBFIND Intrinsic (continued)

DBFIND(base, set, mode, status, item, argument)

Mode	Argument	Description
22	Same as mode 11	Same as mode 11, but does not return qualifying count.
23	Same as mode 12	Same as mode 12, but does not return qualifying count.
100 200 300		Qualifies whole datasset, starting at the beginning of the key.
400 500		Qualifies whole dataset, starting at the end of the key.
1nn	Partial key value	Qualifies all records equal to (=) the value, evaluating nn nimber of words, or -nn number of bytes
2nn	Partial key value	Qualifies all records greater than (>) the value, evaluating nn number of words, or -nn number of bytes.
3nn	Partial key value	Qualifies all records greater than or equal to (>=) the value, evaluating nn number of words, or -nn number of bytes.
4nn	Partial key value	Qualifies all records less than (<) the value, evaluating nn number of words, or -nn number of bytes.
5nn	Partial key value	Qualifies all records less than or equal to (<=) the value, evaluating nn number of words, or -nn number of bytes.

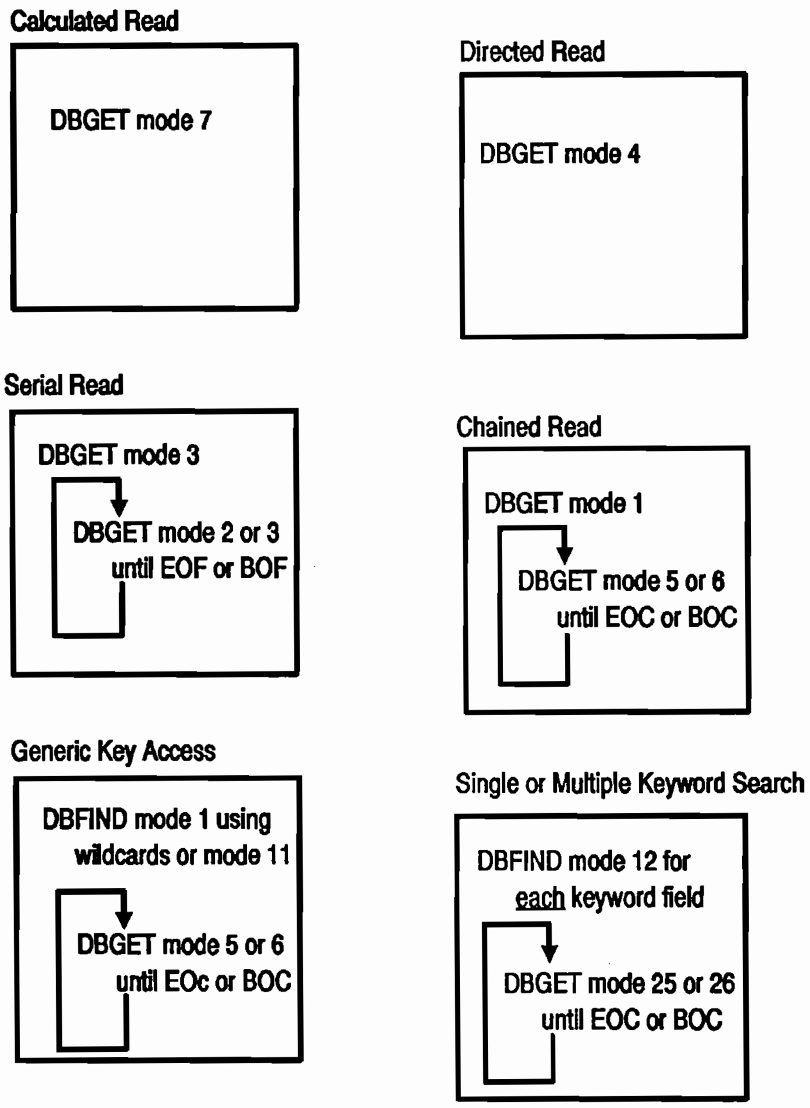
Figure 2 - New DBGET Intrinsic

(Note: This is not a complete list of modes and argument tokens. Please consult the Omnidex manual for a complete list.)

DBGET(base, set, mode, status, list, buffer, argument)

Mode	Argument	Description
5		Read next record in sorted order using generic key access.
6		Read previous record in sorted order using generic key access.
11		Reset to beginning of sorted working set using generic key access.
12	Number of records	Move forward specific number of records in sorted order using generic key access.
13	Number of records	Move backward specific number of records in sorted order using generic key access.
15		Read next record in sorted order, even if beyond the end of the working set using generic key access.
16		Read previous record in sorted order, even if before the beginning of the working set using generic key access.
21		Reset to beginning of qualified working set using multiple keyword retrieval.
22	Number of records	Move forward specific number of records in qualified set using multiple keyword retrieval.
23	Number of records	Move backward specific number of records in qualified set using multiple keyword retrieval.
25		Read next record in qualified set using multiple keyword retrieval.
26		Read previous record in qualified set using multiple keyword retrieval.

Figure 3 - Retrieval Constructs for TurboIMAGE/iX


Calculated Read
DBGET mode 7
Directed Read
DBGET mode 4
Serial Read
DBGET mode 3
DBGET mode 2 or 3 until EOF or BOF
Chained Read
DBGET mode 1
DBGET mode 5 or 6 until EOC or BOC
Generic Key Access
DBFIND mode 1 using wildcards or mode 11
DBGET mode 5 or 6 until EOc or BOC
Single or Multiple Keyword Search
DBFIND mode 12 for each keyword field
DBGET mode 25 or 26 until EOC or BOC

#3159

THE EXPANDED IMAGE RETRIEVAL TOOL KIT: A GUIDE TO USING IMAGE AND THIRD PARTY INDEXING ACCESS METHODS

Donna Bloechl
Dynamic Information Systems Corporation
652 Bair Island Road, Suite 101
Redwood City, California 94063
415/367-9696

At the 1990 Interex conference in Boston, Hewlett-Packard announced that it would provide a direct interface within TurboImage/XL to third party indexing packages. This interface is expected to be implemented in the 3rd quarter of 1992. It will provide the long awaited generic key capability, as well as sorted sequential retrieval, keyword searches, and multiple key searches using standard Image intrinsics with a purchased third party indexing product. Given these new capabilities, this seems like an opportune time to review the Image access methods that have always been available and to introduce those that will be provided with the new third party interface.

SERIAL

Description:

1. All non-empty records read in forward (DBGET mode 2) or backward (DBGET mode 3) order.

2. In a detail set, all records are examined read up to the high watermark.

3. For a master, all records are examined.

Advantages:

1. No key values needed.

2. Can use any field as selection criteria.

Disadvantages:

1. Almost always too slow for online applications. (Your users will curse your name!)

Performance tips:

1. This is the fastest method if you are after a large percentage of the dataset. How large? More than 10% is a very general guesstimate. This will actually vary depending on several factors. On MPE/V, the break even point when comparing serial read speeds with keyed reads is around 1/BF (BF = Blocking Factor). In other words, if the number of records you are retrieving is more than 1/BF in proportion to the file size, a serial read will yield better performance than keyed reads. So if the blocking factor of a dataset is 3, 1/3 the capacity (for a master) or 1/3 of the high watermark (for a detail) is the "break even point". For MPE/iX, there are several variables that will determine whether keyed reads or serial reads will yield the best results. These factors include CPU size, memory size, and current system load. With enough memory "elbow room", MPE/XL will do forward serial reads with blazing speed. My best advice is benchmark!

2. On MPE/V, a large blocking factor will favor serial read performance. But be careful not to make your block sizes too large or your online performance will suffer (for a complete discussion of this topic, see Bob Green's chapter in <u>The Image/3000 Handbook</u> called "Block that Buffer").

3. Details should be periodically repacked using your favorite database utility in order to reset the high watermark to the current capacity.

4. Master sets that are read serially should not have excessive empty space since a serial search will examine all records, including the empty ones. This advice must be balanced with the concern to leave enough empty space to allow for growth and to minimize performance degradation due to secondaries. On masters with hashed keys (type U,X,Z, or P), the number of records in the set should not exceed 70%.

<u>DIRECTED</u>

Description:

A single detail record is retrieved using the relative record number.

Advantages:

1. Speed! This is the fastest access method available.

Disadvantages:

1. Not recommended for masters due to migrating secondaries.

2. You must know the relative record number.

CHAINED

Description:

All detail entries with an identical key value are read in forward or backward order. The chain can be kept in chronological or sorted order.

Advantages:

1. Good performance if one is after all records in the chain or selected records from a relatively short chain.

Disadvantages:

1. Must know entire key value. Partial (generic) keys not allowed.

2. No relational operators (>, >=, <, <=) allowed.

3. No Boolean operators (AND, OR, NOT) allowed.

4. A maximum of 16 keys is allowed per detail set, but only one can be used for a chained search. One cannot select on a combination of keys (for example, all the orders for product XYZ placed in November).

Performance tips:

1. Select search items that have a large range of values. An example of a bad choice would be a flag with only 2 possible values. Having to traverse a chain consisting of 50% of the records in a detail set will result in angry users beating a path to your door! Even in batch mode, a serial read of the set will probably be faster than chained access if the chain exceeds 10% of the total number of records.

2. Repacking the set using a DBUNLOAD,CHAINED/DBLOAD or a database maintenance utility will optimize the performance of chained retrievals using the search item chosen for repacking.

The DBUNLOAD,CHAINED will use the primary path, but database maintenance utilities will allow you to choose. The search item should be carefully chosen based on two criteria:

a. The search item most frequently used for online chained access; and
b. The search item with the longest chains.

3. Keep in mind that synonym problems in a linked master will affect the DBFIND performance on the detail.

<u>CALCULATED</u>

Description:
Records from masters are retrieved using the search item value.

Advantages:
1. Very fast, unless your set suffers from problems with secondaries (see Performance Tips below).

Disadvantages:
1. Only one key allowed per master.

2. Must know entire key value. Partial (generic) keys not allowed.

3. No relational operators (>, > =, <, < =) allowed.

4. No Boolean operators (AND, OR, NOT) allowed.

Performance tips:
1. The primary (no pun intended) concern with master sets is to minimize performance degradation due to secondaries.

   a. Don't allow the number of entries in a master to exceed 70% of the set capacity.

   b. HP has long recommended that the master set capacity be a prime number. While the effectiveness of this has been hotly debated by many, it is easy to implement and certainly won't cause any harm.

   c. Monitor secondaries with an appropriate database performance utility (such as HowMessy).

2. Non-hashed keys (type I, J, K, or R) have excellent performance **if** you educate yourself on their care and feeding. In the hands of the uninformed, a previously well-behaved master set can be a performance time bomb that can turn ugly overnight. Fred White's articles on the subject are required reading if you plan to utilize these.

NEW TOOLS PROVIDED BY THIRD PARTY INDEXES:

The four Image access methods discussed so far--serial, directed, chained and calculated--are available with the Image package from Hewlett-Packard. The following two, generic and keyword, are available with the purchase of a third party indexing package. The features offered by third party indexing vendors will vary. The features described here are all part of the standard minimum set of features that all third party vendors agree to provide. Vendors may choose to provide additional features and enhancements beyond this standard.

GENERIC

Description:
Generic key access provides the ability to use wildcards and relational operators in a search argument. All qualifying records will be retrieved in sorted (by key value) order. This will be accomplished with a DBFIND mode 1 using any of the following operators and wildcards:

@	Any number of letters or digits
?	Any single letter or digit
#	Any single digit
>	Greater than
> =	Greater than or equal to
<	Less than
< =	Less than or equal to

After qualifying entries with a generic argument, DBGET modes 5 and 6 are used to retrieve the entries in ascending (mode 5) or descending (mode 6) sorted order.

This is only a partial description intended to give you some idea of the functionality of the generic key. For further details on additional modes and correct syntax, please refer to your indexing vendor's documentation.

Advantages:

1. Use of wildcards allows much greater flexibility. One example would be the ability to easily retrieve all records that have an area code of 510 (`510@`) or a zip code range (`940@`).

2. The ability to retrieve records in sorted order can eliminate sorting operations and thus greatly speed up batch reports when selecting 10% or less of the records in a dataset. This feature also makes online reporting a viable option.

3. Selection and sort fields may be combined to form composite keys. A generic composite key can replace the functionality of a sort field in a detail without the performance degradation problems.

4. Multiple generic keys may be added to masters or details. This, in effect, eliminates the Image limit of one key per master and 16 keys per detail. Now you can actually make a logical master a physical master and still have flexible access by multiple keys!

5. The value of a generic key may be changed with a DBUPDATE. Image will now allow this to be done on an Image search item in a detail but not on a key in a master. The ability to change a key using a DBUPDATE, instead of having to perform a DBDELETE and DBPUT, results in less overhead.

Disadvantages:

1. One cannot do several DBFINDs using generic key retrieval in succession So even though you may have ORDER-DATE and PRODUCT-NO defined as generic keys in a set, you cannot search on all the product XYZs that were purchased in May. This search may, however, be accomplished by installing a composite key which combines these two fields.

2. No Boolean operators allowed.

<u>KEYWORD</u>

Description:

1. Records are qualified based on individual words or values contained in fields. For example "`DATA`" can be used to qualify all company names such as "`WESTERN DATA COMPANY`" or "`XYZ DATA`".

2. One or several keywords can be used to qualify on a single field. For example, "WEST@ COMP@" will qualify company names "WESTERN COMPUTING" and "COMPUTERS OF THE WEST".

3. All the wildcards and relational operators (@, ?, #, >, >=, <, <=) listed under Generic key access are also available with Keyword retrievals.

4. Boolean operators (AND, OR, NOT) may be used (for example: BOB OR ROBERT).

5. Multiple keywords can be used to qualify records based on values in several fields. For example, you could find all the customers who are located in California who ordered product XYZ between March and June of a given year.

6. Multiple keyword search criteria need not all be located in a single data set.

7. Keyword searches are accomplished by doing a DBFIND mode 12 for each keyword field, followed by DBGET mode 25 (forward) or mode 26 (backward).

Advantages:

1. The ability to retrieve based on any word or value in a field is very useful for text fields such as product or part descriptions. For example, a user can search on all product descriptions containing the words "COTTON" and "SWEATER".

2. Qualification of records based on multiple keywords is done by accessing the index files and using very few I/Os. No I/O is done on the datasets until all qualification is complete. For example, let's say we want to select all the customers located in California who ordered product XYZ between the months of April and June, and that the following records qualify:

   ```
   20,004  STATE = CA
   14,887  PRODUCT = XYZ
    8,991  ORDER-DATE = 9104@ OR 9105@ OR 9106@
      247  Meet all 3 criteria
   ```

   I/O would be done only on the 247 customers that meet all 3 of the search criteria.

3. Multiple keyword keys may be added to masters or details. As with generic keys, this makes it possible to have more than the Image limit of 1 key per master and 16 keys per detail.

4. Use of Boolean operators allow great flexibility. For example it's handy to be able to search for `ROBERT OR BOB OR BOBBY OR ROBBY`.

5. Fields may be combined to form composite keys.

6. Selection may be done on ranges of values in a field. This can be especially useful for date ranges.

7. Each DBFIND mode 12 will return the qualifying count. Quite often this may be the only thing that a user is after. A sales manager may be interested in knowing how many wigits were sold by salesman Barney from January through June, but likely will not care to see the details of each sale. The qualifying count is returned very quickly using very few I/Os.

8. The value of a keyword key may be changed with a DBUPDATE. Image will now allow this to be done on an Image search item in a detail but not on a key in a master. The ability to change a key using a DBUPDATE, instead of having to perform a DBDELETE and DBPUT results in less overhead.

Disadvantages:

1. Once users get used to the flexibility of keyword retrieval, there is a great temptation to get carried away. The overhead is paid on DBPUT, DBDELETE, and DBUPDATE. A key that is used pays for itself and is easy to justify. A key that is rarely or never user used ("We may need it someday!") needs to be reconsidered.

<u>SUMMARY:</u>

Before the addition of the third party indexing interface, Image provided four access methods:

1. serial
2. directed
3. chained
4. calculated

A third party indexing package used with the new Image interface will provide two more options:

1. generic
2. keyword

With these new tools, there is a great deal of flexibility available using Image. There is no one type of access that will be the best method in all situations. Each tool has advantages and disadvantages. It is important for the database administrator to know the retrieval needs of the users in order to provide the access that will give the most flexible access and the best performance. These enhancements make Image more robust than ever and provide all the search capabilities necessary for today's business users.

ACKNOWLEDGEMENTS:

Sincere thanks for technical editing and advice provided by Steven Cooper of Allegro Consultants and Eric Savage of Dynamic Information Systems Corporation.

Retrieval Constructs for TurboIMAGE/XL

Calculated Read

```
DBGET mode 7
```

Directed Read

```
DBGET mode 4
```

Serial Read

```
DBCLOSE mode 3

   v
   DBGET mode 2 or 3
     until EOF or BOF
```

Chained Read

```
DBFIND mode 1

   v
   DBGET mode 5 or 6
     until EOC or BOC
```

Generic Key Access

```
DBFIND mode 1 using
  wildcards or mode 11

   v
   DBGET mode 5 or 6
     until EOC or BOC
```

Single or Multiple Keyword Searc

```
DBFIND mode 12 for
  each keyword field

   v
   DBGET mode 25 or 26
     until EOC or BOC
```

PAPER NO.: 3160

TITLE: Moving From TurboIMAGE to Relational

AUTHOR: Greg Proulx
Oracle Corp.
500 Oracle Parkway
M/S 659-408
Redwood Shores, CA 94065
(415) 506-6162

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO.:	3162
TITLE:	Data Integrity From TurboIMAGE To Relational
AUTHOR:	Jo-Ning Ta Oracle Corp 400 Oracle Parkway Redwood Shores, CA 94065 (415) 506-7200

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO.: 3163

TITLE: Understanding Relational DBMS

AUTHOR: Jim Eduljee
Infocentre Corporation
10600 W. Higgins Road
Suite 703
Rosemont, IL 60018
(708) 803-0450

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

Paper number: 3166

TurboIMAGE/XL 4.0: Relational Access

Bradmark Technologies, Inc.
4265 San Felipe Suite 800
Houston, Texas 77027
(713) 621-2808

Tim Joseph,
Technical Support Manager
Larry Boyd,
Research and Development Product Manager

Introduction

The value of IMAGE is a function of its ability to retrieve data. External Third Party Indexing Products currently provide enhanced data retrieval solutions. At the 1990 INTEREX conference in Boston, Hewlett Packard announced the direct interface from TurboIMAGE to existing third party indexing products. Indexing Technology will have been integrated within the TurboIMAGE/XL intrinsics when version 4.0 of TurboIMAGE/XL is released (3rd quarter '92).

This paper discusses:

a) indexing technology improvements facilitated by the <u>operating system level interface</u> between TurboIMAGE/XL version 4.0 and Indexing Technology;

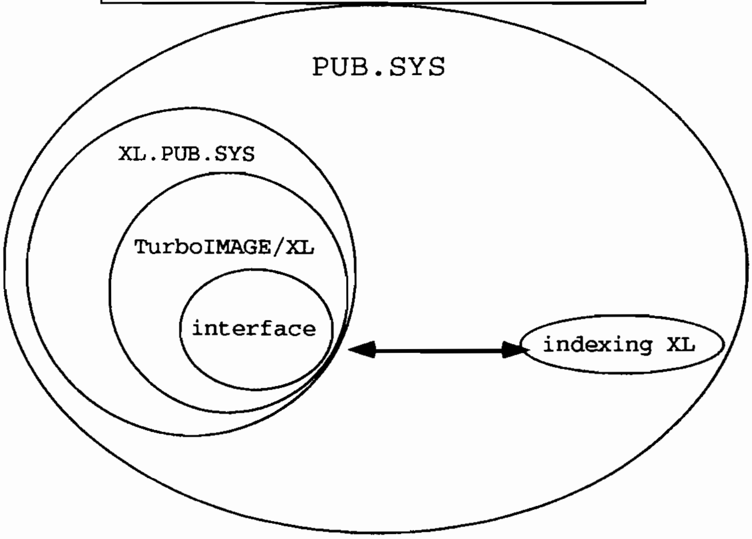
b) the <u>simplicity of indexing</u>; and

c) the increased value of more accessible data via the relational access capabilities of TurboIMAGE/XL's enhanced DBFIND intrinsic.

Operating system level interface

With TurboIMAGE/XL 4.0, indexing will be handled directly within the intrinsics. This provides several major benefits. Index synchronization is enforced by enabling the database for indexing via DBUTIL. Transaction logging, locking, and system failure recovery are integrated in a seamless fashion. Indexing maintains Binary-tree indexes in hidden standalone detail datasets for which IMAGE does not perform Image Transaction logging. IMAGE locking strategies and TurboIMAGE/XL Transaction Manager (XM) include index management. For example the XM steps are: 1) XM transaction starts; 2) IMAGE modifications; 3) Index modifications; and 4) XM transaction completes; therefore, when the database is accessed after a system failure, Transaction Manager insures the integrity of data and indexes.

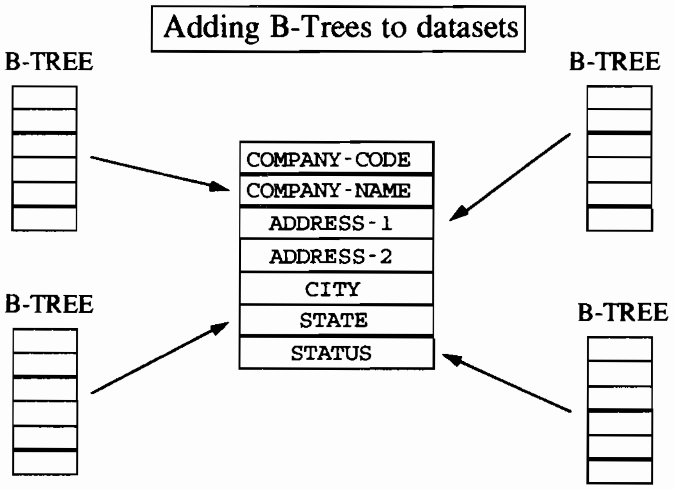
Operating system intrinsic level interface



A direct interface between TurboIMAGE/XL and the indexing XL, which contains the indexing routines, has been created.

B-trees

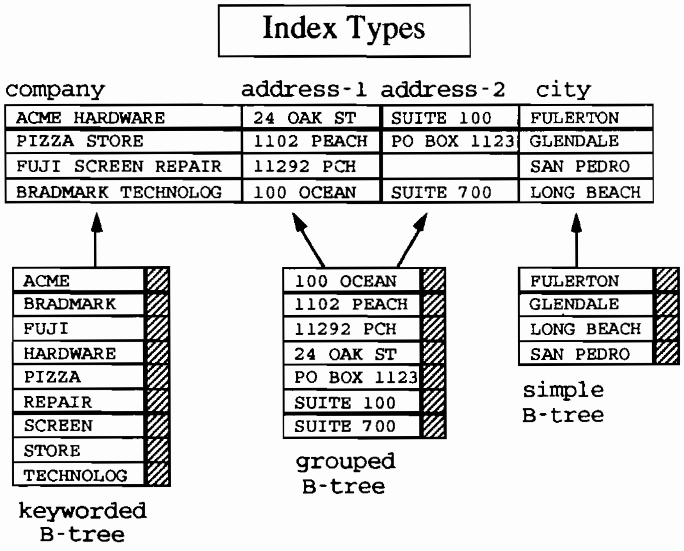
Binary-trees are easily configured and attached to data items in datasets. Once an indexing structure is defined, its synchronization is guaranteed: as records are modified, the indexes are maintained through TurboIMAGE/XL.



Every B-tree generated is identical in structure: a sorted list of keys with a pointer. The question is: What in the record is worth indexing? Indexing allows you to be flexible when specifying which data item, or items, or portions of items in a dataset become keys and what type of keys they become.

B-tree Key Types

These are a few of the types of indexes which can be generated. Note that the differences result from how the data in the IMAGE items are converted into B-tree key values, not the type of B-trees.



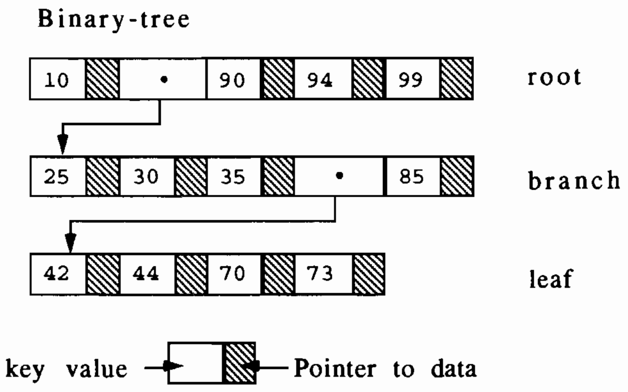
The dataset above contains 4 entries. The keyworded B-tree contains 9 keys. For example, Fuji Screen Repair is referenced in the B-tree by `FUJI`, `SCREEN`, and `REPAIR`. The grouped B-tree contains 7 keys, two for each entry (minus the blank entry which by default is not indexed). The simple B-tree contains 4 entries.

Indexes can be combined. For example, a grouped keyworded index can be useful. The above example of address could be keyworded grouped instead of just grouped.

B-trees: structure

This example shows the basic structure of a B-tree. A compression algorithm minimizes disc space usage. Hidden datasets contain the B-trees. These datasets are not seen by `QUERY> form sets`. When IMAGE logging is enabled, no logging is done for hidden sets. `DBSTORE/DBRESTOR` tapes will contain the hidden datasets and their B-trees.

Binary-tree

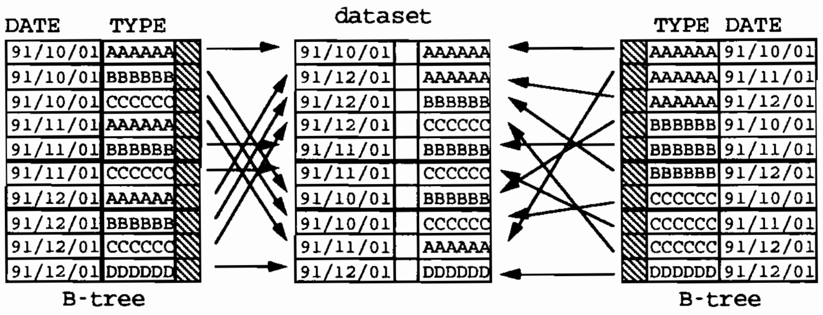


The B-tree consists of the key value(s) from each entry in the dataset being indexed and a pointer to that entry. For detail datasets the pointer is the relative record number of the entry in the dataset. Master datasets use the full IMAGE key item as a pointer because entries in master datasets can change their relative record number during normal operations (due to migrating secondaries).

B-tree = sorted order

The B-tree maintains the index key value in sorted order. The pointer allows the entries in the dataset to be read. Indexing uses the sorted key values in the B-tree to qualify entries and then gets the data using the pointer. For master datasets, the extension is used as the argument in a `DBGET mode 7` (calculated get) and for detail dataset the extension is used as the argument to a `DBGET mode 4` (directed read).

Binary-tree SORTED pointers to the dataset

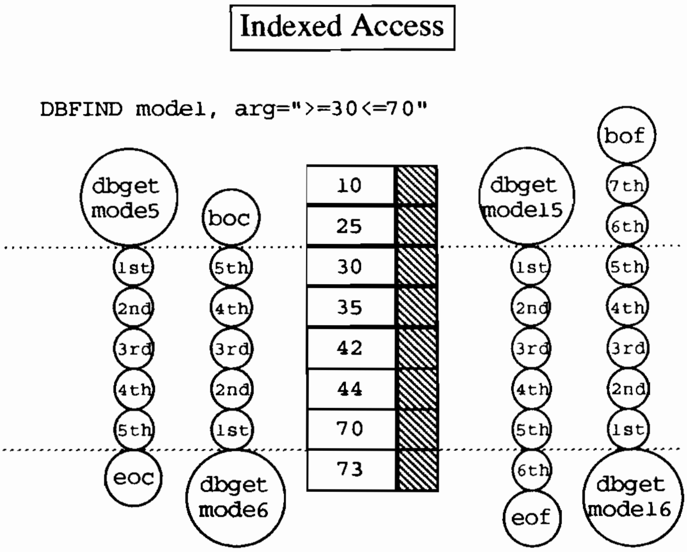


This sorted ordering can be particularly useful with concatenated keys. For example, if `DATE` and `TYPE` are concatenated, the entries would be sorted as shown above in the left B-tree. The B-tree on the right shows the ordering for another key with `TYPE` and `DATE` concatenated. Multiple sorting orders can exist simultaneously.

DBFIND mode 1

When a DBFIND is called with an ITEM parameter set to an index, a chain of qualifying entries is selected. Pointers are set to the first and last qualifying entries in the B-tree and the number of qualifying entries is returned in the IMAGE status words 5 and 6 (chain count).

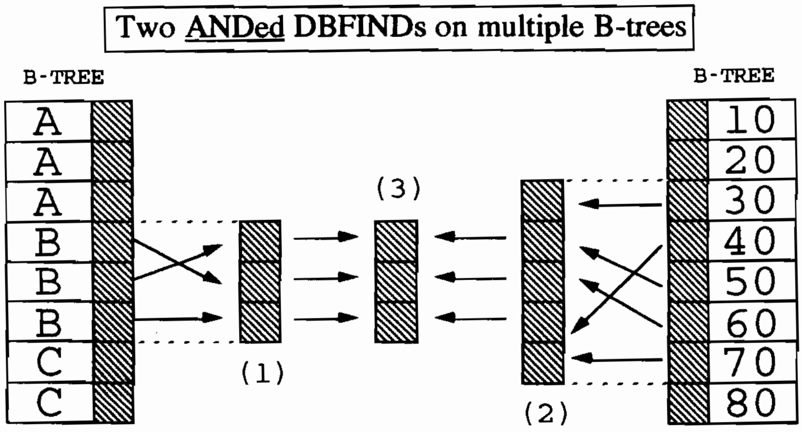
The argument may contain:
'@', '?', '#', '>=' or '<=' and combinations thereof.



If the DBFIND is followed by a DBGET mode 5, the "chain" of qualifying indexes is read in ascending order until END-OF-CHAIN is returned. If the mode is 6, the chain is read in descending order until BEG-OF-CHAIN is returned. If the DBGET mode is 15, the chained read is not inhibited by the normal END-OF-CHAIN and continues on reading in ascending sorted order until END-OF-FILE is reached. Likewise, if the DBGET mode is 16, the chained read is not inhibited by the normal BEG-OF-CHAIN and continues reading in descending sorted order until BEG-OF-FILE is reached.

Two ANDed DBFIND mode 12s on multiple B-trees

Below multiple DBFINDs are performed against multiple indexing B-trees. The first DBFIND has an argument of "B;" creating the relational list (1). The second DBFIND is called with an argument of "AND >=30<=70;". This causes the relational list (2) to be created based on the argument ">=30<=70;". Then the "AND" is performed between (1) and (2), giving a relational list (3) which represents all entries meeting both criteria.

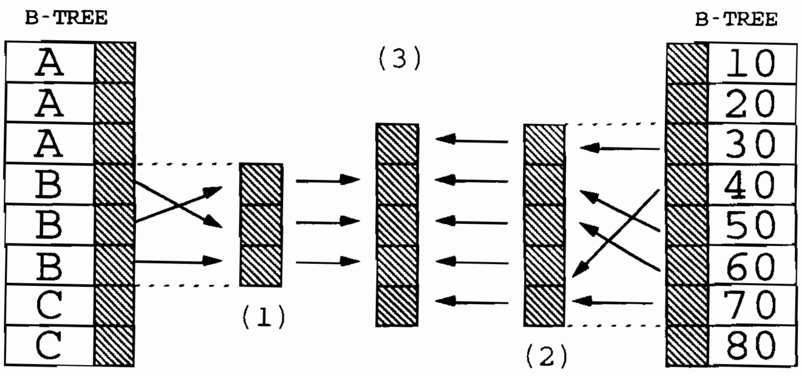


Imagine the code required to accomplish the same task without an indexing structure. A chained read is performed, checking which records in the chain meet the second criteria. When those records to be selected can be specified directly with boolean relationships, selection logic is localized in one section of code instead of being spread between the DBFIND and a filter on the DBGET (with considerable unused data moved to the user's program stack/heap).

Two ORed DBFIND mode 12s on multiple B-trees

Below multiple DBFINDs are performed against multiple indexing B-trees. The first DBFIND has an argument of "B;" and creates relational list (1). The second DBFIND is called with an argument of "OR >=30<=70;". This causes the relational list (2) to be created based on the argument ">=30<=70;". Then the "OR" is performed between (1) and (2), giving a relational list (3) which represents all entries meeting both criteria.

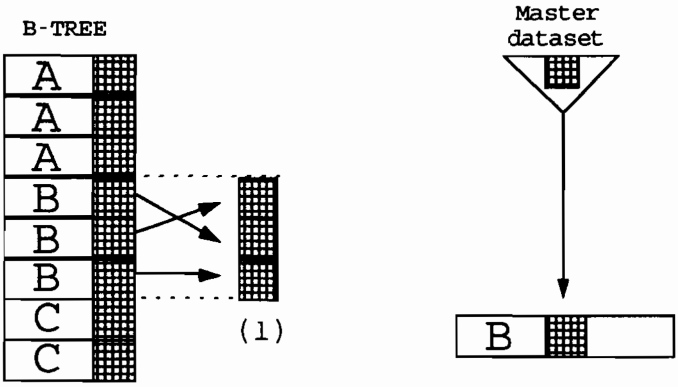
Two ORed DBFINDs on multiple B-trees



Two related datasets, DBFIND on one B-tree

Indexing allows the grouping of one or more master sets with one or more of their related (by IMAGE paths) detail sets. Master entries may be qualified based on the contents of the related detail entries. The contents of the detail become the index key and the pointer is to the related master dataset. Here the pointer is the related master dataset's key value as found in the detail datasets data item.

B-tree with detail's key, pointer to related master

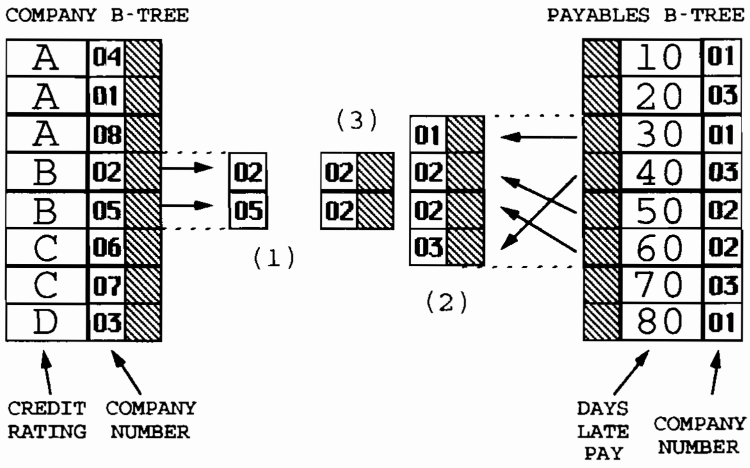


In this example, a DBFIND searches the B-tree for all detail entries with a value of "B;". The list (1) contains the master search item from the details where "B;" was found and can be used to read the qualifying entries in the master dataset.

two datasets, DBFINDs on multiple B-trees

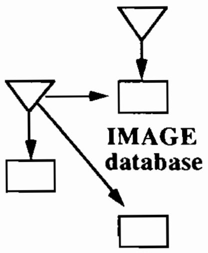
When doing boolean operations between relational lists, a relational item is needed. When the lists are from the same dataset, the relational item is the relative-record-number/image-key-value for detail/master datasets. When more than one dataset is involved the relational item must be specified in the DBFIND.

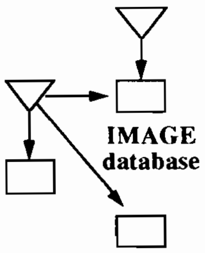
Two ANDed DBFINDs on separate datasets



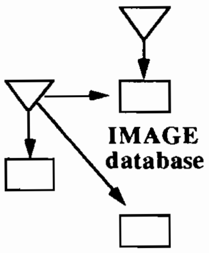
The first DBFIND has an argument of "B;" with the relational item specified as COMPANY-NUMBER and creates the relational list (1). The second DBFIND is called with an argument of "AND >=30<=60;" with the same relational item as the first DBFIND. This causes the relational list (2) to be created based on the argument ">=30<=60;". Then the "AND" is performed between (1) and (2), giving a relational list (3) which represents all entries which meet both criteria and can be related via the relational item. Now a DBGET mode 5 causes the relational chain to be traversed and the pointer retrieves entries in the payables dataset where days late are ">=30<=60;" and whose company number related credit rating is "B;" in the customer dataset.

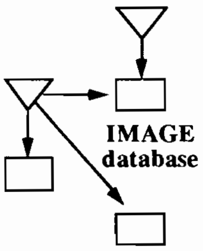
Two databases, DBFINDs on multiple indexes





The solution for linking datasets in two databases is the same as used for two datasets in the same database. The multiple database scenario is viewed as a special case of the multiple dataset retrieval where it just happens that the second dataset is in a different database.



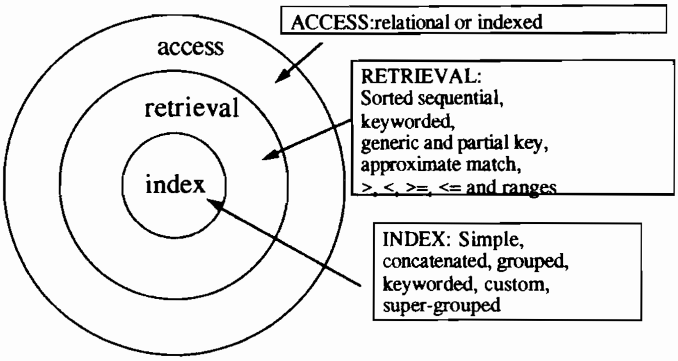


Simplicity of Indexing

Indexing is simple because it is modular. Each piece is independent of the others. An index can be defined without regard to the argument or retrieval method.

The types of INDEXes are: simple, concatenated, grouped, keyworded, custom, super-grouped, substring or a combination of types. These types specify the KEY values which will be sorted by the B-tree.

Index Modularity



The **retrieval** argument of `DBFIND` can be varied to search in the desired manner: 1) Partial key search: `UNIT@`; or 2) Generic key search: `UNIT???`; or 3) range retrieval: `">=911000<=911299"`; or 4) approximate match.

The Access modes are Indexed and Relational. Indexed access refers to one argument directly accessing the B-tree. These types of searches do not involve boolean operators as in the date range: `">=910101<=911031"`. You can retrieve the first entry of January, then the next, in sorted order, until you reach the end of January. Relational access allows multiple arguments to be related via the boolean operators: `AND`, `OR`, `NOT`. For example, everything sold by Fred related to all open receivables by "AND" gives the result of open receivables sold by salesman Fred.

Conclusion

With the increased accessibility to data provide by IMAGE via indexing, the value of the data has increased. But the increased value is only realized if the new features are implemented in applications. When the perceived benefit is large enough and the cost and complexity small enough, INDEXING should gain widespread usage. It would seem reasonable to replace KSAM files whose sole purpose has been to provide what IMAGE did not but now does: PARTIAL KEY lookup. Where only a fraction of the entries in a dataset are qualified during a serial read, INDEXING can be utilized to reduce wall time enormously (for example many QUIZ™ reports).

It might be the case that user requests not previously feasible, are now possible using the new DBFIND modes. Where otherwise sound user requests were unfulfilled due to limitations on data retrieval, perhaps it is time to reconsider. The IMAGE product enhancement list has the most requested item, partial and generic retrieval, marked completed via the direct interface from TurboIMAGE to existing third party indexing products. Powerhouse™ QUICK applications are just a few statements away from making more from existing screens. The unused potential of possible retrievals is considerable and must be considered by 3rd party application vendors when deciding what enhancements to implement. Previously, user community retrieval enhancement requests might be deemed to difficult to implement or deflected with rhetoric about there not being an accepted indexing standard; but now there is a standard.

In almost all situations end users can benefit from INDEXING. Some estimates (they may have been our own) indicate that in five years:
a) no one will be using TurboIMAGE;
b) most applications will have been migrated to ALLBASE; or
c) 80% of all IMAGE applications will utilize INDEXING; or
d) none of the above.
Survey says: C) 80% of all IMAGE applications will utilize INDEXING! Accept no substitutes!

Normalizing IMAGE In The 90's

By

Rick Hoover

Introduction

A lot has been written about relational data base normalization. There are 5 forms of relational data normalization. Do these only work on relational data bases? That is the question I started with for this paper. The thought that only relational data bases were able to be 'normalized' did not seem logical. After all, a data base is a data base, right? There are differences between the structure of a relational data base and an IMAGE data base. BUT, can the topics covered in the 5 forms be defined for IMAGE?

The first step I took was to look at the main structure difference between IMAGE and relational data bases. I have worked with a few relational data bases on a PC and I have worked with IMAGE for almost 15 years. The differences are not as great as first thought. On first glance, the most glaring difference was the type of safety that IMAGE has given us. Relational data bases contain tables. Each table is a stand-alone file containing records. These records are the same type as any IMAGE data set.In fact, by replacing the word data set for table and there is no difference. Well, almost.

Each table in a relational data base can be accessed by a program. The records in a table may be edited, added or deleted. So far, no difference. We can add, edit or delete any record in an IMAGE data base. WRONG! Because of the structure of a relational data base we can do things in a relational data base that we cannot do in an IMAGE data base. For example, if you have a data set (table) that contains a purchase order header record and a purchase order detail record, you can delete any purchase order header record in a relational data base but, if the purchase order header is in a manual master data set and attached to one or more purchase order details, you cannot delete the header. IMAGE has already made the linkage between the two data sets. Without the correct programming, a relational data base will allow the deletion of the header. This is referred to as an 'anomaly' in relational data bases. This term doesn't really crop up in IMAGE.

Now, back to the question of relating (no pun intended) a relational data base back to IMAGE. Can IMAGE look and feel like a relational data base? Well, almost. I attended a talk many years ago on IMAGE. This talk came back to me because, possibly unknowingly, the speaker defined a way of having a look and feel IMAGE relational data base. He suggested that all data sets become details with automatic masters attaching the linkages between sets. This way you can have multiple paths (up to 16) to any set, master or detail. So, with that in mind let's lay out some terms.

The terms between relational data bases and IMAGE are pretty close. A data set in IMAGE is a table in a relational data base. (Note: for the people out there that have purchased a PC package called ALPHA4 will note that ALPHA4 refers to each table as a data base and a collection of data bases as a data set!)

Each record in an IMAGE data base can be called a row in a relational data base. Each field (or entry) in IMAGE is a column in a relational data base.

Form 1 - Eliminate Repeating Groups

Example: A purchase order header record contains a 3 line message area of 60 characters each. These 3 fields show up on the header screen for the user to type in some standard information for the purchase order. These 3 lines were designed to comply with the 'worst case' scenario as defined by the users. These 3 lines are optional.

This example (as well as all other examples used in this paper) are from actual experience. This message area was defined in the schema as PO-MESSAGE 3X60.

All 3 occurrences of this field were defined as optional. In a relational data base (at least on most PC based systems) space is allocated as each record is added. That means that as each record is added to a table, about 180 bytes are allocated whether used or not. In an IMAGE data base the space is allocated at data base creation time. If the data set requirement is set for 10,000 records then the data set will take up an additional 1,800,000 bytes of disk space. I'll guarantee you that most of that space will probably not even be used.

By moving the message fields out of the purchase order header data set and placing the data in its own data set will accomplish not only satisfying the first normal form but also a few other things. Disk space will be returned to the system. I realize that one argument is that disk space is cheap. But so are MIPS. Access time has gotten so fast that the argument becomes mute.

Of course you have also locked in the total number of lines of messages for the purchase order. Granted, the user base may have said, on a stack of Bibles, signed in blood and offered the first born that this will NEVER EVER change. Don't take the bet. Usually within 2 months of implementation a user will come up and say "Well...we have this one instance that REALLY REALLY need 4 lines". You can always get around this situation with a detail data set containing just the messages. You can't get around this with a 3X60 type field.

There are two notes I would like to offer about setting up these kinds of data sets. First, these don't work with all array type of fields. However, I would like to offer a suggestion about array type fields.

I had a data base that contained a field that contained all the costs for an item. It was defined COST-FIELD 12P12. The first 4 occurrences contained the labor, material, standard inventory valuation and cost of sale costs for current costs. The second 4 contained the above for frozen costs and the last 4 contained all the above for the new costs. This does not truly apply to a separate data set. But, to remember which is which does become a problem. Remember, when designing a data base, it's easy to know which is which, but after 6 months or a year, the knowledge is lost.

The second situation is keeping the data in the correct sequence. Please don't throw stones at me. The easiest way is to place a sequence number on the data set and place a (gulp) sorted path on the data set. I can already see the panic set in. Sorted paths have been given a bad name over the years. I agree that some applications of sorted paths are wrong. If you can lock in a limitation on the number of transactions on a path (say, under 50) then a sorted path is pretty safe. Otherwise, I would recommend a sequence number and have the program do an internal sort.

Form 2 - Eliminate Redundant Data

Example: A purchase order has a field for the buyers name on the header record defined as BUYER X20. The user can type in the name of the buyer in this field. This field is required.

This field creates redundant data in that the user is required to type in the name every time a new purchase order is created. This assumes that the user community will type in the buyer name the same way every time. Can you imagine seeing a report that sorts the open orders by buyer with a page break on buyer name. One data entry person may put in John Jones, another will put in Jonathan Jones, another may use Jones, J and the last may use Jones, John. Cross referencing this on line becomes a nightmare. The buyer will never find all the orders assigned.

A better way would be for the user to key in a buyer code and let a data set containing the cross reference from buyer code to buyer name take care of the situation. This solves the form 2 problem. It also solves a typical problem that plagues data bases. That problem is the constant moving of employees from one task to another, or employees leaving the company. It is so much easier to assign a code to each employee. This way, if John Smith, (buyer code 14) moves to another department and Mary Jones takes over the spot, the users only need to change the name associated with the buyer code to reassign all buyer information. If the name is placed in all records, the data processing staff will need to get involved to change the name. Of course, with relational data bases, the search through all tables can result in missing one or two transactions.

There is one 'gotcha' in this scenario. There is a company that is eliminating redundant data in their data bases by using codes. However, this also refers to other objects. Do not assume that you are successfully eliminating redundant data by getting rid of the data in a data base. You must not put the data in a program either. This company has work centers defined with codes but the codes are defined in programs. The same holds true for planners and routings. This creates a mess when a planner leaves or a work center changes. The planner code exists in over 12 programs. The programmer has to go into each of the programs to make the change.

The work center information is even worse. The work center information is stored in an MPE file. BUT, there are many programs that access this MPE file. How do they access the MPE file, you may ask. Well...each program opens the MPE file and reads in all records into a table in the program. They have not had the capability to fix this program because there are so many programs that use the work center information. They know that there is a maximum of 100 table entries. BUT...they are not sure how many programs use this file. So, if they need 101 work centers...well they can't. They have had to make it policy not to go over 100 work centers.

Form 3 - Eliminate Fields Not Dependent On The Key

A different way of saying this is to not have any other data in the data set having duplicate information. This normalization statement goes along with form 2. The data base designer should create other keyed sets to contain the basic information necessary to validate the data. The main difference between this form and form 2 is that any field that further defines information that does not purposely describe the main keyed item should be moved to another data set. It also allows linkages through one set to another. For example, a requisition data set is generated. A purchase order is created from the requisition. The requisitioners number is on the requisition and the purchase order contains the requisition number. By not placing the requisitioners number on the purchase order you have insured consistency. You can get the number by going through the purchase order. There is probably a data set that contains the requisitioners number and name.

At this point you probably have the makings of a solid normalized data base. The structure is solid for most of the concepts of normalization. But, like anything else, there are some steps that can make the data base structure even tighter, more well defined. By adding in the last two forms of data base normalization, you will achieve the yin and yang of data base normalization (Impressive, huh?!).

Form 4 - Isolate Independent Multiple Relationships

This design applies to only data sets that contain one to many or many to many relationships. For example, we have an item master that contains a lot of data about a specific item number. We wish to know if the item can be made of steel and if the item can be sold to a distributor. These two attributes do not share a meaningful relationship, therefore they do not qualify for the normalization process because the item may also be made of steel and be used for export. This does not allow for the item to be loaded into the data base. The data base should be built to allow a separate data set for the distribution type linked to the item number. In fact you could take the item makeup (plastic, steel etc.) and create a data set for this information. This allows the flexibility to have an item that can be made of different materials and be available for different distribution methods.

I can hear the mind working through this one. And, I state again, as the computer systems keep getting faster with shorter access time for disk reads, there should be no reason for a programmer (or data base designer) to NOT allow as much flexibility in a data base right from the start. Designing a data base for today without thinking about tomorrow is one of the worst design modes that we in the industry have. (I will now get off my soap box, thank you)

Form 5 - Isolate Semantically Related Multiple Relationships

This type of normalization can be a bit confusing, especially when dealing with an IMAGE data base. Let's suppose that you have a data set that contains a vendor code with a vendors supplier and the type of supply they offer as in:

Vendor	Supplier	Type Of Material
101	1001	White Plastic
101	1001	Blue Plastic
101	1001	Red Plastic
101	1002	White Plastic
101	1002	Blue Plastic
101	1002	Red Plastic
101	1003	White Plastic
101	1003	Blue Plastic
101	1003	Red Plastic

As you can see by the above table you will need 9 entries to contain all the necessary data. This table does support the first 4 normalizations of a data base. To add a new vendor with this type of information would be a lot of data entry. To create the fifth normal form we need to create 3 new data sets as shown:

Vendor	**Supplier**
101	1001
101	1002
101	1003
Vendor	**Type Of Material**
101	White Plastic
101	Blue Plastic
101	Red Plastic

Now, instead of having to add nine new transactions, we only need to add 6. This type of arrangement becomes more difficult to quickly create. There is a lot more thinking about creation of the cross reference of this normalization than the other 4.

In conclusion I would just like to state that data base normalization can work for ANY kind of data base, not just relational ones. By applying the rules of data base normalization to your data base structures will help increase throughput and maintenance. Of course, the more analysis done on a data base structure, the better! I would not just want look at the outside of a house and say that is the house I want. I would first check out the neighborhood, the structure of the house, access to stores and schools, etc. You should go through the same thought process before releasing a data base on the user community. After all, you may have to live with the data base for many years to come!

Paper Number 3201

Systems Integration vs Open Systems

The Road to Enterprise-Wide Data Sharing

By

Eric B. Threatt

Wesson, Taylor, Wells & Associates

Information Technology Consulting

P.O. Box 23587

Columbia, S.C. 29224

(803)-699-5781

As companies seek increased competitiveness and cost reduction through enterprise-wide "data sharing", they are faced with a complex dilemma. A move to open systems seems to be the way of the future, but what about the current investment in proprietary hardware and software? Should a company's investment in proprietary systems be suddenly abandoned, especially in uncertain economic times?

Integration of current systems is a possibility. But these systems were never designed for "seamless" communication and interoperability.

Though the true adoption of open systems in the commercial business environment is in the early stages, the improvements in efficiency and long-term cost reduction cannot be ignored.

Most companies agree that to maintain competitiveness, enterprise-wide data sharing is of vital importance. Given the current state of technology, the choices seem to be systems integration, open systems or some combination of these options.

The advantages and disadvantages of the move to open systems must be evaluated carefully, given the need to balance long-term efficiencies against short-term economic realities. The role of systems integration must also be considered with respect to it's contribution to the achievement of the company's information goals.

In Search of Open Systems

A concrete definition of open systems has been difficult for many companies to obtain due to the different "spin" that computer firms are putting on the open systems concept. Some computer companies have described open systems simply as systems that utilize UNIX. Other companies, though espousing commitment to open systems, approach this concept through interconnection of a number of their own proprietary systems.

Widely accepted definitions of open systems coincide with compliance to standards such as the Open Systems Interconnection (OSI) Reference Model, developed by the International Standards Organization (ISO) and the Systems Network Architecture (SNA) developed by IBM.

The OSI Reference Model is comprised of seven "layers". The first three layers of the reference model define how the connection will be established ensuring that data can be passed from one point to another in virtually any format. Layer 1 describes the physical connection between systems (including connections, cabling, etc.), while Layer 2 (Datalink Layer) and Layer 3 (Network Layer) are responsible for error correction and routing of data from origin to destination, respectively.

Layers 4 through 6 (Transport, Session and Presentation) are primarily responsible for the formatting of data. Layer 7, or the Application Layer ensures that the right screens and formats appear, even though the information may have come from a different system than the one being utilized by the user.

The Systems Network Architecture (SNA), is similar to the OSI model, though there are some differences. For example, the definition of the layers are different as is some of the functionality. Some of the SNA protocols are being actively considered for adoption by OSI.

These standards specify how intelligent devices exchange information. Much of the interest in compliance to these standards stems from the increased flexibility and reduction in costs that will result from establishment of an open environment.

Some computer firms such as Hewlett-Packard began participating in OSI early. In 1983, HP became the first major computer company to publicly commit to phasing out proprietary networks in favor of OSI.

Compliance to standards, combined with commercial quality UNIX, are often cited as forming the core of open systems. Commercial UNIX, often referred to as a more robust form of UNIX, provides a number of features not commonly for in the versions of UNIX historically employed by the scientific community.

Though the concept of "openness" does not mandate the employment of UNIX, the commercial definition seems to utilize UNIX and "openness" as synonymous terms. From an application perspective, the idea of an "openness" standard seems to be settling around POSIX in a P/A RISC (Reduced Instruction Set Computing) environment.

POSIX is a standard developed by the Institute of Electrical and Electronics Engineers (IEEE). The POSIX approach focuses on the development of a standard interface between application programs and operating systems. RISC is the result of a simplification of instructions used to process information, and a dramatic reduction in the number of these instructions. The development of RISC has resulted in significant improvements in processing efficiency and a reduction in the cost of computing.

There are four primary features that outline the attractive nature of a move to open systems. The first benefit is the portability of software. The ability to move a software application from one open system to another is, of course extremely appealing.

The scalability of the open architecture allows for the movement of applications between open computer systems, regardless of system size. This helps a company protect it's software investment regardless of the size of computer that might be required by the company in the future.

There is increased availability of software between systems in an open systems environment. And finally, interoperability, or the ability to effectively network these systems maximizes efficiency.

The move to open systems has the potential to be slow and painful. There is much support for the concept, and many companies are taking the first steps to open systems through applications such as electronic mail.

There is, however, wide spread apprehension about a deliberate moves in the direction of open systems. In spite of the advantages, many MIS managers are not at this point convinced that the return is worth the risk.

The financial managers in the company are also concerned about the investment that they have already made in the proprietary systems. There are concerns about training costs and, most importantly, concerns about major disruption in the company's ability to transact business.

All of these issues are worthy of consideration when evaluating an approach to enterprise-wide data sharing. Some of the fears outlined above result from a lack of information and education about what is now proven technology. Some of the costs and disruptions can be controlled with effective planning and management.

However, even if a company is made to feel more comfortable about the technologies employed in open systems, this still may not dispel the question about and discomfort with the short term cost of abandoning thousands and potentially millions of dollars worth of hardware and software investment. Are there alternatives?

Systems Integration

As a remedy for these short term cost considerations, the growing trend seems to be in the area of systems integration. The goal in systems integration is, in essence, the same as the open systems goal in terms of practical application; the integration of the company's information resources to improve the efficiency, responsiveness and competitiveness of the company.

The problem in the systems integration approach is that many of the proprietary systems were never designed to communicate "seamlessly" with one another. Contemporary networking approaches improve the odds for success.

However, this is not a substitute for the interoperability and portability of systems under the open systems concept. Still, depending on the state of the company's current technology, it's financial constraints and specific needs, the use of the systems integration approach may, in the short term, be a more comfortable solution.

As many companies will opt in the short term for the integration of the proprietary hardware that they currently possess, a prudent approach to this process is clearly advisable. The process must be managed professionally by a steering committee and systems and business professionals that have identified and documented the dynamics of the project or projects, and have formulated strategies that will help ensure the success of the integration effort.

Reduced Pain through Effective Planning

The first step on this path in the development of a enterprise-wide information technology plan. Many feel that this plan must begin with the technology. Nothing could be farther from the truth.

The existence of the information systems environment is based on it's ability to contribute to business goals of the company through information. Clear identification of the company's business goals and how information facilitates those goals is a critical first step. From this step, integration projects will be not only defined, but also prioritized.

However, keep in mind that a company's business environment is constantly changing. The key to a successful systems integration plan, like any plan, is it's flexibility in the face of a changing business environment.

Another important part of the planning process is to ensure that the planning team is comprised of members that understand the business goals as well as the technical goals. There is a direct correlation between the quality of the planning at the beginning of the process and the pain (or lack thereof) experienced during and after completion.

Moving Toward The "Openness" Goal

The long range goal of the company must be to achieve "openness" in the pursuit of enterprise-wide data sharing. The reality of a move to open systems for the majority of companies lies with the understanding of and commitment to the importance of development of an open systems environment, and a phased approach to getting there.

There are three primary features that the long-range information technology plan must focus upon to ensure that enterprise-wide data sharing is achieved. First, the company's strategic business goals and what information supports those goals must be identified.

Secondly, required data and the uses for that data must be identified so that redundancy can be eliminated and efficiency maximized. And thirdly, as the company moves toward an open environment, it is important to mandate the any newly acquired hardware and software should conform to the open systems standards.

After these steps are completed, a parallel approach to enterprise-wide data sharing evolves. New systems that are acquired will adhere to open standards. With the identification of critical data requirements, identification and prioritization of systems integration projects can begin.

Long Term Goals and Short Term Realities

Commitment to open systems and the integration of current proprietary systems are, in the long term, somewhat mutually exclusive in concept. The reality is that in the short term, both can be pursued in tandem, as long as the company is focused on openness as the long term goal.

While it is impractical to assume that companies will simply throw out their proprietary systems, the move toward open systems can begin with an understanding of the gains to be derived. A plan for "getting there" will include systems integration projects to facilitate enterprise-wide data sharing, and conformance to standards with new development and acquisition.

There are a number of tools and technologies today that can bridge the integration-openness gap, providing the first steps on the road to enterprise-wide data sharing. For example, proven technologies that provide data transport between proprietary systems are available today.

Migration from large proprietary systems to smaller open systems can be less painful and expensive through the use of migration tools that port current applications to the new system.

The plan must be carefully constructed and executed, with an eventual move away from proprietary architectures in mind. It is this balanced approach that will ensure that the company meets it's long term goals of integration and interoperability.

3202 OPEN SYSTMS

David W. Haberamn
Nicholas Walsh
Mary Kaminski
Scott Becker
Linda Haake

Speedware Corporatiom
One Penn Plaza
Suite 1624
New York, NY 10119
(212) 563-2145

If you looked at any computer publication over the last five years you would find two words over and over again throughout the articles. Thise two words have become the MIS batrtlecry of the 90's. Every MIS department wants to have them, Every hardware vendor claims have them, and every software vendor says they have them. however if you ask any two groups what those two words mean you will get a different answer. In this paper we will look at those to words and how they effect your current and future MIS envirionments. Those two words are OPEN SYSTEMS.

Studying the concept of Open systems is like wading through a bowl of alphabet soup. There are more Acronyms used in describing open systems than the military used in describing the last three wars. In this paper we will review the five major areas that effect open systems and define the acronyms that help describe those areas.

The five areas we will cover will be

Database Systems (DBS)
Graphical User Interfaces (GUI)
Object Oriented technology and Prgramming (OOT/OOP)
Computer Aided System Engineering (CASE)
Client Server Technology

We will also take a look at why open system are so important in todays environment. We will review this from a historical perspective as well as current environments and future directions.

What are open systems. These words mean many things to many people. To start off with we will offer this definition:

Open systems are systems that transparently access information and resources trough a multivendor network of hardware platforms, operating systems, and database management systems that look and behave as an integrated whole.

The concepts and reasons for open systems are as old as data processing itself. To begin analyzing open systems we must first look at the history of computers and data processing.

HISTORY

The first computers were single use systems. A computer would be built to perfom a certain function or set of functions. These systems had their programs

built right into the hardware. The input to these systems was either by punched card or wiring changes.

Programs were written by changing the wiring. The output from these systemns were reports. Each system was a unique entity. Data was stored in a format unique to the system. Wiring changes were not standard from system to system.

These systems were very expensive. The largest users of these systems were governments and insurance companies. Whole systems were developed just to compute shell trajectories. These systems were very expensive to build and maintain and were not very flexible as to what they could do.

To make systems more flexible, computers that could accept instructions from outside sources were developed. These systems accepted instructions (programs) from punched cards. The instructions were in machine code. The machine code was unique from system to system. Output was either to tape or to report.

Systems were still unique to a department or organization. One computer would be used to meet the needs of the whole organization or individual computers would be bought to meet departmental needs. Organizations began to realize a need to share data. To accomplish this standards in data storage formats were developed. ASCII (American Standard Code for Information Interchange) and EBCDIC (Extended Binary Coded Decimal Interchange Code) became the standards for storing and accepting information.

During this same time the need for more efficiant computer programming languages became apparent. The need to develop computer instructions in a quick and easy manner became a prioroty. The US government, the largest users of computers was insisting that its vendors develop and adhere to standards. In the 50s, 60s, and 70s a number of computer programming languages were developed. Many of them were standardized by ANSI (American National Standards Instituet). Different languages were developed to meet diffent needs. Over 170 different languages were developed.

COBOL (Common Business Oriented Language) became one of the defacto standards of the business world. In and age of punched cards, reports, flat files, and tape drives, COBOL was an open system. COBOL could be taken from one system to another by changing just the environment section of the program. This was true of other languages also. ANSI helped organizations standardize their languages.

The economics of the situation was that standardization of systems allowed organizations to get the most from their training dollar and port systems from one platform to another. This helped save money in system maintanance and development.

However advances in computer technology kept coming. Different vendors would add functionality to their languages. This functionality would enhance the environment but decrease portability. Every few years ANSI would look at all the innovatuions and set a new standard.

Innovations kept on coming up with technology not covered by the standards. Terminals were developed for data input. This meant that instructions had to be included to accept input from a terminal. CRTs were developed and instructions had to be developed to write informatoin to the screen.

One of The biggest innovations was database technology. New ways to format and retrieve data were being developed. Each database vendor required a diffent set of instructions to access their data. The systems were no longer portable.

During this period Bell laboratories was trying to develop a development environment to meet their growing needs. This environment became UNIX. Unix was developed as a flexable, easy to use, and portable operating system. The goal was one operating system and development environment to meet all of Bells needs. Out of the same development project Cwas created. Future versions of UNIX were written C.
C was hailed as the new standard in programming. Its portability and flexibility made it the language of choice for many organizations with UNIX the operating

system.

UNIX was subject to the same limits as other environments. Although highly portable it lacked some functionality. To increase functionality vendors released their version of UNIX. ATT UNIX, SCO XENIX, SCO UNIX, AIX, ULTRIX, and HP-UX are all versions of this operating system. If the user wrote systems that took advantage of the enhancements provided by a vendor then the system was no longer portable. If you wrote systems that only used the "Lowest Common denominator" of UNIX then those systems weren't as efficiant.

During the 70s the price of computer hardawre started to drop considerably. Organizations were realizing that the ability to manage information woould help them become more competative. As hardware prices dropped individual departments in organizations were buying systems to meet departmental needs. No consideration was givenm to sharing information with the organization as a whole. Decentralization was the business buzzword at the time and computers followed suit.

In the 80s computing power came to the individual. PCs (Personal Computers) brought information to the desktop. Organizations realized they had a wealth of information on their systems but no way to share organization wide. Organizations needed ways to share data.

As hardware costs decreased software development costs increased. Organizations needed the ability to cut down on development time and make the most use of their systems and programming resources. 4GLs (Fourth Generation Languages) and code generators started to evolve. These innovations speeded up development. The problem was these languages were not portable and many times were tied to a proprietary database. Vendors actually developed different versions of their language for each hardware platform.

The old standards did not meet the needs of the organization. Computer networks, EDI, and communications protocals were developed. Vendors were coming up with innovations faster than standards could keep up. The standard became that of whichever vendor could grab the majority of the market place. organoizations started to realize that they needed more than defacto standards. The investment in information processing within an organization was so significant that the need for industry standardization had become obviuos.

The term OPEN SYSTEMS was born.

When interviewed 55% of data processing managers said they would use open systems technology if it interfaced with current systems. 40% said systems based on open systems technology were a prioroty over the next three years.

As organizations invest money in systems they want to be assured these systems will meet their needs in the future. In this day of mergers, divestures, international economy, and electronic transactions. The need for open systems is greater than ever.

What are open systems? Many people equate open systems with UNIX. UNIX may be a component of open systems but is not all inclusive of open systems. We gave you one definition already. Here are two more.

X-OPEN is a consortium that includes 95% of all hardware and software vendors by volume. This group is dedicated to defining open operating systems, file structures, and communications. X-OPEN defines open systems as:

Software environments consisting of products and technologies which are designed and implemented in accordance with standards established and defacto that are vendor independant and that are commonly available.

The Gartner Group defines open systems as:

A set of standard APIs, not yet complete, addressing everything from operating system services to datamanagement to system management to the workstation user interface.

The word standards is featured in each of these definitions. There are two types of standards defacto and established. Eestablished standards are developed by commitees or consortium and are adhered to by the industry as a whole. Defacto standards are established by whichever company manages to get the majority of the market share in its field.

Apple and Tandy both came out with personnal computers before IBM. However IBM set the defacto PC standard. CPM was a PC operating system before MS-DOS. However MS-DOS is the defacto standard. Communication standards for EDI were established by committe and adopted by the government. This made them the standard. Hayes compatible has become the standard for modem communication.

However how do you handle innovation in a world of standards? Suppose we set our standard for user interfaces and tomorrow someone develops holographic voice acuated moniters? How do we fit this into the standard? What about video or voice interfaces? Remeber are example of COBOL. Each COBOL vendor would add some enhancements to go with their system. Later these enhancements would be included in the standard. By the time a standard is decided upon has the technology already surpassed it?

We have given you just a few examples of standards and the issues involved. We will now review some of these standards and the terms used to describe them.

Graphical User Interface, commonly called GUI, is one of many hot topics in the computer world today. Everyone is talking about them and most people want them, except for the true "techies" , who are trying to cling to their ever familiar, and old friend, the dos prompt.

GUI is a generic term applied to user interfaces that incorp-orate layered windows, pull-down menu bars, picture "icons" that represent programs or activities, and/or color. GUIs are aimed at making tasks as easy as selecting a picture to accomplish the needed task. By eliminating the need to remember cumbersome and forgetful key-stroke combinations, GUIs are making word processing easier, to name one of many applications that are using GUI.

One might find oneself overwelmed with the colors, the menus, and the icons, but once inside the application you begin to see the "intuitiveness" behind this interface. Instead of looking at your template across your function keys, and your color coded "shift key" and "ctl key", you simply choose a picture of a printer to print your current document. So, as the initial shock wears off, you'll find yourself zipping through your task quite rapidly without having to remember any more key-strokes.

There are a few well known GUIs out in the market place at this time. Microsoft Windows 3.0 is a product that extends and enhances the DOS operating system with the use of layered windows and graphical icons. Presentation Manager (PM) is an OS/2 GUI that follows IBM's System Application Architec-ture(SAA). SAA is IBM's blueprint on computing technology and the standards of the technology that would permit computers to work together in a consistent fashion. X-Windows is a basic windowing system developed by MIT (Massachusetts Institute of technology) for UNIX/POSIX environments. The foundation for Motif. Motif is a GUI designed for UNIX/POSIX environments that is used on HP9000's and is expected to be available on HP3000's as well. Also known as OSF/Motif.

Another buzz word in the market place today is OOT/OOP.

Object Oriented Technology and Programming are among the latest hot topic of the industry today. This paper is going to give an overview of the concepts behind OOT, and define the eight components that OOT is based upon.

To define OOT, it is a theory of application or system design that treats all the parts of a system as "objects", whether they are data entities, data definitions, programs documentation, or processes. The goal behind OOT is to make the managing of more complex systems easier, improve programmer productivity, create more flexible systems, and make maintenance less of an issue.

Object-oriented technology enables you to represent more closely the model of your business in the computer. There are eight concepts that OOT is based on. The first of these is an object, an object can be anything around you. An object can be a shoe, book, person, etc. Objects have attributes associated with them i.e. you might want to know a person's name, number, and address. Objects also have behaviors associated with them such as payroll, 401k plan.

Secondly, we have class, class is a template or shell for defining data relationships and tasks assigned to a collection of similar objects.

Third, is inheritance which allows subclasses to utilize tasks and data formats from their superclass(es).

Fourth, is instance, an instance is an object that is associated with a particular class. An instance has a specific value. So an instance can be thought of as a record in a file.

Fifth, is encapsulation. Encapsulation is a technique in which data and tasks are packaged together. This occurs at the object level.

Sixth, is message. A message is a request sent from one object to another object to perform a specific task.

Seventh, is method. A method is a task that can be performed on or by an object.

Eighth, is polymorphism. Polymorphism literally means "many forms". It's the ability to send the same message to different objects and get results based on the method defined for the class of the receiving object.

Now that we have defined the components of OOT, we can briefly address what this offers us. By having a centralized repostory where the objects are stored, we can easily browse/modify existing objects. Reusing objects will increase programmer productivity and make maintenance a simpler task, while making software more reliable, since your using an existing, tested object. Ojects isolate the program functions from each other, so there is little or no ramification to other sections of application.

This portion of the paper will briefly define the traditional DBMS's and two of the newer concepts in data base systems.

Traditionally there have been three common database models, these being hierarchical, network and relational. The first two being good at modeling complex hierarchies of data, but you must begin at the root and follow a path specified by your program. Ad hoc reporting off of this data without predefining a path is impossible. If you want or need to change a link, you must manually modify the schema and your application to utilize these changes. You must also close access to your data for this to be accomplished. So we have a good method for quick retrieval of data, but it was a rigid inflexible method.

The relational database offers us flexible data retrieval, but it is less effecient. The concept behind RDBMS's is to think of data in a table with rows and columns. The standardization of the access language, SQL (Structured Query Language), was a major step forward for relational technology. Once you learn SQL on one platform it's easy to move to another platform. SQL has three parts to it.

1. Data Manipulation Statements
   SELECT
   INSERT,UPDATE and DELETE

2. Data Definition Statements
   CREATE
   ALTER
   DROP

3. Control Statements

GRANT and REVOKE
COMMIT and ROLLBACK
LOCK

Relational databases don't store the relationships between the tables in the database. This is accomplished at runtime using a report writer or programming language.

The folowing is a list of relational highlights taken from Orly Larson's Abstract "Information Management Technologies in the 90's".

- relational concepts are easy to understand and use
- SQL is a multifunctional language
  * Database definition and creation
    Data retrieval and manipulation

  * Authorization and security
  * Transaction Management and recovery
  * Database environment management and restructuring
  * Interactive and programmatic use
- Data Independence is insured and minimizes program maintenance
- Data access is automatically optimized as Db structure changes
- the DBA has unprecedented power and control over the D B
- Standard access to data
- User specifies what information wanted, not how to retrieve it
- Increases programmer productivity, lifts programming to the level of problem-solving
- Provides for SQL tools and application portability
- Assist in cross-system connectivity
- Basis for a true distributed database environment
- Faster implementation of new systems

Some of the leading relational databases today are Oracle, by Oracle, Sybase, and HP's Allbase.

As we move forward in technology to meet business needs, two other database types exist, they are Distributed Data Base and Object Oriented database. The first being a concept defined two ways by Chris Date.

1. "A DDB system is a system involving multiple sites connected together in a communications network. In which Each site is a DB system site in its own right plus A user at any site can

access any data in the network exactly as if the data were all stored at the user's own site.

Thus, a DDb is a virtual DB whose components are physically stored in a number of distinct REAL DBs at a number of distinct sites".

2. "A DDb system is a system that allows:

   - an arbitrary collection of relations
   - from an arbitrary collection of databases
   - on a variety of different machines
   - running a variety of different operating systems
   - connected by a variety of different communication networks

   to function as if they wre all stored in a single database on a single machine.

   The user is completely insulated from all details of the distribution".

By these two definitions, we can see that this a concept and not actually a single type of database. It's the rising need of accessing data from various types of existing databases at local and remote sites. We won't go any further into to detail at this point, since this is addressed more specifically in the client server portion of the paper.

The second of these is Object Oriented Databases which use the concepts described in the OOT portion of the paper. ODBMS expands on the relational database by supporting both predefined access and user-defined methods(tasks). This type of DBMS allows you to combine tasks and data, allowing users the flexibility to model most real world business information .

Generally speaking, ODBMS allow more information to be managed by the database in a shared environment, thus freeing the programmer from mundane coding, which leads to faster development time. We now have data and code(tasks) residing in the database to be shared by users.

Today with the RISC architecture, performance is not a major issue with relational databases and more shops are going to this structure. It's predicted that by the end of 1992 that 65% of applications will be developed using an RDBMS compared to only 7% in 1988.

CASE or Computer Aided Software Engineering refers to the discipline adhered to during the process of developing applications. Methodologies like CASE ensure the quality of of the final application by identifying each step in the development of an application and formalizing the standards of which to adhere. CASE addresses one of the most critical business needs today: the ability to quickly and effectively manage a business' information assets.

A formal definition shows that CASE refers to the Tools and Methods that increase the quality of applications and decrease the investment necessary to develop and maintain them.

CASE Tools aid and automate the process of design, implementation and maintenance of applications. They support every phase of applications development including planning and analysis, design, implementation, testing, and maintenance. CASE Tools range from PC-based graphical design tools to server-based version control and archiving tools.

The ideal CASE system will move the information from the endusers' requirements to an executable system automatically. All information in the process will be maintained in the repository, and accurate documentation will be automatically produced.

A useful way of categorizing CASE products is to distinquish between the employment of the tools at different life-cycle phases. In so doing, we come up with UPPER CASE Tools and LOWER CASE Tools.

Upper CASE tools address planning, analysis, and design phases. This means that they interface with both the endusers and the analysts, and are critical in getting the system development started correctly. The key to the greatest useability of Upper CASE tools is their capability of transporting or transforming data; between for example, the repository and the tool. It is fundamental that the Upper CASE Tool selected to work with have free communication with the central repository.

The central information repository is the critical and essential element of any integrated set of CASE tools. It is the source of information for code generation and the mechanism for code reusablility. It is the database for storing and organizing all the components of an application system.

Lower CASE Tools, on the other hand, address implementation, testing, and maintenance. Although Upper CASE Tools often run on PC's to take advantage of graphics, Lower CASE Tools are often run on a server or multiuser system.

Some examples of Upper CASE tools are system planning, data flow diagrams, and object oriented design. Examples of Lower CASE tools are source coding tools, use of reusable code, integrated word processing, and presentation graphics.

The main advantage of Computer Aided Software Engineering lies in the ability to easily deploy and modify information systems. CASE provides this advantage by reducing the investment required to build and maintain new applications.

The features of CASE that are generally considered to be the most desirable are support for the entire application lifecycle, use of an information repository, and support for basic programming techniques.

CASE addresses the growing maintenance backlog of MIS departments. Up to approximately 75% of MIS resources are devoted to the maintenance of existing applications. This inhibits the ability of businesses to develop new information systems and increases the cost of operations. CASE Tools can alleviate much of the coding burden of programmers thereby increasing the speed of implementation and maintenance, and decreasing the likelihood of errors.

The term, Cooperative Processing, is the big "buzzword" in data processing today. Although we hear it everyday, many people are not quite sure what it means. In short, cooperative processing is the ability to process data across CPU's regardless of where data lays. When we hear the term today, it is usually applied to the ability to integrate PC or workstation processing power into the mainframe or minicomputer processing environment.

This processing across platforms is also commonly referred to as Client/Server environment. It usually refers to an environment in which designated CPU's "serve out" software and data to client CPU's and peripherals. The server CPU is designated as a "traffic cop" to route information and data. The client systems manage the data processing. However, client server can also describe the process of one software process serving out data and/or processing to another software process.

Some examples of cooperative processing environments we see everday include point of sale systems, lottery ticket sales, and airline reservation systems.

A few acronyms that fit in with this topic include:

DCE: Distributed Computing Environment is a technology that consists of a full set of descriptions that relate to file access, remote procedure calls, user environments, and networking. (DCE's provide a very promising interoperability tool.)

RCP: Remote Procedure Calls, which are supported by DCE, are high-level facilities that allow programs to call one another without concern over the different processors they run on or the languages they were written in.

NFS: Network File Systems, also supported by DCE, give us the ability to access files at the record level on remote computer systems.

Historically, the advent of computers gave us the ability to process and store large amounts of data. This allowed us to perform mission critical processing on an organizations core data. There were problems with this. All data was maintained at a centralized location. Access was limited to those with a direct link to the mainframe. If you were in another city, you needed a leased line and 24 hour a day connections to process remotely. In many environments if the system went down, everyone was down. For example, if you did order entry at a terminal and the system went down, order entry stopped until the system was back online.

In the last ten years we have seen a great shift in data processing. The processing power in PC's and workstations is increasing to compete with minicomputers. The price of hardware and disk space is coming down. These two factors are making it cost effective to put data processing power on the individual desktop.

As this technology advances we are seeing a major trend towards data and communication standards. By standardizing the ways we store, access, and transmit data, we are developing the ability to make processing transparent across systems.

There are many advantages to cooperative processing. The ability to process data locally provides speed and accessibility to all users. The processing power needed to handle graphical user interfaces(GUI), screen editting, and storage of data is available in PC's and workstations. By offloading these functions to PC's, you reduce the amount of processing power and storage space needed on the mainframe or minicomputer.

In many applications the PC or workstations can continue to process data even when the minicimputer or mainframe is offline. In statisitical information gathering it may not be necessary to store all the gathered data. The PC or workstation could gather the detail information, process it, and then transmit the results to the minicomputer or mainframe for analysis by management.

In an order entry environment the standard process in a cooperative processing environment may take an order, check the past customer info on the mainframe, compute the order, and reserve the item in inventory. The actual order is maintained on the PC and updated in batch at the end of the day. If the mainframe is down, order entry can still occur.

As MIS professionals we all know that the technology for cooperative processing is available today. We have also seen many models of cooperative processing environments.

What are some of the issues we face in establshing a cooperative processing environment?

1. Database Access

As HP3000 users we need access to our IMAGE data. Most of our current systems have IMAGE data bases. Whichever platform we look at must access that data base platform. We also need access to advancing technologies such as relational data base management systems.

2. Development Environment

We do not want to have to train our people on multiple languages. The ability to use one language and methodology across platforms will help reduce training, development, and software maintenance costs.

3. Interface with Futute Technologies

As new technologies develop we want the ability to take advantage of them. Systems that do not take emerging standards into account may become outdated very quickly.

Conclusion

These are the emerging technologies for open systems today. As MIS professionals look to purchase and develop systems we must work with the tried and true technology of today but build systems flexible enough to include the technologies of tommorrow.

Paper # 3251

Using Powerhouse For Data Migration

Marilyn Petty Glick
Time Systems, Inc.
5353 North 16th Street
Suite 400
Phoenix, AZ 85016
(602)265-3220

Introduction

Time Systems' business is training people to use their time more efficiently. This is done with workshops that teach how to use Time Systems' Time Activate System. Time Systems also produces the forms and sells the supplies required. Time Systems recently converted from a PRIME computer to an HP3000 Series 948 running Distribution Resources Corporation's (DRC) System For Distributors (SFD).

Although DRC provided some programs to migrate data to the new system, they did not address all the data that Time Systems needed to migrate. I would like to share our experience and the Powerhouse techniques used, in hopes of making data conversion a little less painful for any shop embarking on such a project.

Environment

Time Systems had a PRIME computer running order processing, inventory control, projections (a forecasting system), purchasing and financials. As the company grew, the PRIME system no longer met the company's needs and the decision was made (March 1991) to acquire SFD and a new HP3000 Series 948. By May, the new machine was installed and the new application was loaded. Now, you must understand Time Systems' business involves calendars. People think about new calendars toward the end of the year thereby creating Time Systems' "busy season". Since the PRIME would not handle another "busy season", the new system would have to be up and running by September at the latest. Moreover, needed custom features would also have to be ready, the data moved, and people trained for the new software. In house, we had one PRIME person (who also knew HP), one HP programmer, one operator who had never worked with computers and one operations manager, new to HP and also managing the PC LAN. Our deadlines were tight - so we added two contractors with lots of experience to our staff - and got started.

Data

Almost all of our data was on the PRIME in "MIDAS" files, a type of data base used only on PRIME computers. Our product master had been put into a dBASE file to be more easily changed for the new system.

DRC provided programs to load customers, vendors, ship to's, open accounts receivables and products from ascii flat files and provided specifications for these files. The idea was to write COBOL programs to pull the data from the "MIDAS" files and put it into ascii flat files. Then move these files to the HP3000. The DRC provided programs would work for us, except for the products load. (The PRIME products file was no longer current and not to be moved.)

We would need to write routines to load products, prices, order lines, order headers, alternate order froms, item cross references, extra customer information, order ship tos, component lists and product forecasting data.

Options For Moving the Data

Next, we needed to consider how to get the data from the PRIME to the HP3000. We defined two possible scenarios.

Plan 1:
We could write our ascii flat files to tape on the PRIME and send the tapes to our PRIME support center where the tapes would be copied to reel tapes. These reel tapes could then be sent to DRC to be copied to DAT tapes which could be loaded to our HP3000.

Caveats Plan 1

- Time:	Several weeks to get all this done.
- Complexity:	Too many people/groups involved.
- Mistakes:	None could be accommodated with our tight schedule.
- Expense:	The PRIME Support Center charged for time and materials.

Benefits Plan 1

- Time:	MIS can concentrate on other conversion tasks.
- Certainty:	We knew the data could be moved like this.

Plan 2:
We could download our ascii flat files from the PRIME directly to a PC via VP60 (a PRIME terminal emulator) and then upload them from the PC to the HP3000 using Reflections (an HP3000 terminal emulator).

Caveats Plan 2

- Time:	Download from PRIME to PC is very slow and a PC must be dedicated to that specific task.
- File Size:	Reflections will not upload very large files, it just quits and behaves as if the transfer were successful.
- Hours:	MIS people must be available to watch transfers both ways to be sure all is well and start the next transfer.

Benefits Plan 2

- Control:	Everything can be done in house with existing equipment and regular MIS people.
- Expense:	Less monetary cost.
- Mistakes:	Mistakes/Changes can be accommodated.
- Timing:	No mail/FedEx or outside schedules need to be considered.

Choice

We chose the option of moving the data ourselves because of the control factor, time and cost. This way, we could move the data in pieces; processes could be done in parallel and if we made mistakes, they were ours and we could fix them.

Methodology of Data Migration

The PRIME person wrote COBOL programs to pull the data from the PRIME "MIDAS" files and put it in the proper format to ascii flat files.

These files were then downloaded to a PC hard drive using VP60. Some files took up to eight hours to download.

Reflections was then used to upload the files to the HP3000. Reflections uploaded the files much faster than VP60 downloaded them. We had problems with Reflections when transferring large files (more than 10-15 thousand records) and also with files less than 80 bytes wide. We solved the first problem by limiting the number of records in each file transferred. To get around the second problem, files that were less than 80 bytes wide were changed to be 80 bytes wide. We did not call Walker, Richer and Quinn about these problems as we had adequate work-arounds available. Thus, when a transfer was finished, the file was checked to see that it had the expected number of records and that the proper record length had been specified. If a problem was found, the file was purged and uploaded again.

We used the programs DRC provided to load what they could. In order to get the rest of our flat files into SFD's IMAGE databases, we used Powerhouse. The method that we used is as adaptation of a method Jim Ralph showed me when we were with Northern Telecom.

First, we will do a quick overview of how it works and then go through the conversion of one file in detail as an example.

Overview of Powerhouse Use In Migration

Write a QUIZ routine to access an IMAGE file you will be loading with the ascii flat file. Note that you could load more than one IMAGE dataset with one of your flat files. Define any fields that are not exactly the same as in your flat file. Numbers must be defined as characters and if they are signed, the sign must be defined separately. Set a subfile name and make the size the same or larger than the largest flat file you plan to load with this routine. Make the subfile permanent (keep option). Set the report limit to one. Report all the fields in your flat file in the order they appear in the record. Go. Check to be sure the resulting subfile has the same record length as your flat file and that the file limit is adequate.

Copy your flat file to your new subfile. FCOPY with the option NOUSERLABELS. Using the MPE/XL copy command will cause your subfile to no longer be recognized as a subfile. This is because the file code is changed to the same as the ascii flat file by COPY.

Write a QTP routine that accesses your subfile and outputs to your IMAGE dataset. This routine will change your character numbers and their sign if they have one into numbers for IMAGE. Initialize any fields that are not in your subfile in this routine as well. Be sure your IMAGE dataset has enough capacity to hold the number of records you are going to load. Go.

Example

We will go through the process and code used to add order lines to the ORDLINE dataset of the SFD database.

PRIME File

This is the file description of the file on the PRIME.

```
01  ORDLINE.
    05  ACCTREP                PIC X(2).
    05  COMPANY                PIC X(2).
    05  CUSTOMER               PIC X(6).
    05  DEPT                   PIC X(2).
    05  CHAR-DISCPCT           PIC X(7).
    05  OLD-ITEM               PIC X(4).
    05  LINE                   PIC X(4).
    05  CHAR-PRICE             PIC X(12).
    05  CHAR-QTY-ORD           PIC X(12).
    05  SALES-ORD              PIC X(8).
    05  CHAR-DATE-EXPECT       PIC X(6).
    05  CHAR-DATE-SHIP         PIC X(6).
    05  CHAR-QTY-SHIP          PIC X(12).
    05  CHAR-UNIT-FACT         PIC X(12).
    05  CHAR-EXTENSION         PIC X(12).
    05  CONTRACT               PIC X(2).
    05  CHAR-AMT3              PIC X(12).
```

Notice that all fields are described as character. This is necessary because of the different representations of numbers between the computers.

Download/Upload

The file was filled via a COBOL program on the PRIME. This file was downloaded to the PC using VP60 on the PRIME and then uploaded to the HP3000 using Reflections.

```
ACCOUNT=  SFD          GROUP=  DATA

FILENAME  CODE  ------LOGICAL RECORD-----------  ----SPACE----
                 SIZE  TYP    EOF      LIMIT R/B  SECTORS #X MX

NEWLIN            122B  FA     35732    42000 185       16  1 31
```

QUIZ on the HP3000

Below is the QUIZ routine to make a subfile for order lines.

```
access ORDLINE  <<Access the IMAGE file where the data is going.>>

<<Define the fields in the PRIME file to equivalent fields on the
HP3000.>>
define OLD-ITEM              char*4  = COMP-ITEM-NO[1:4]
define CHAR-PRICE            char*12 = ascii(PRICE)
define CHAR-QTY-ORD          char*12 = ascii(QTY-ORD)
define OLD-SO                char*8  = "        "
define CHAR-DATE-EXPECT      char*6  = ascii(DATE-EXPECT)
define CHAR-DATE-SHIP        char*6  = ascii(DATE-EXPECT)
define CHAR-QTY-SHIP         char*12 = ascii(QTY-SHIP)
define CHAR-UNIT-FACT        char*12 = ascii(UNIT-FACTOR)
define CHAR-EXTENSION        char*12 = ascii(EXTENSION)
define CHAR-AMT3             char*12 = ascii(AMT3)
define CHAR-DISCPCT          char*7  = ascii(DISCPCT)

<<Set subfile size to what is needed to hold the data, keep it.>>
set subfile name ORDLINE keep size 42000

<<Set report limit to 1 as we are not interested in data from here.>>
set report limit 1

<<Report the fields into the subfile the way they appear in the
PRIME file.>>
report summary          &
  ACCTREF               &
  COMPANY               &
  CUSTOMER              &
  DEPT                  &
  CHAR-DISCPCT          &
  OLD-ITEM              &
  LINE                  &
  CHAR-PRICE            &
  CHAR-QTY-ORD          &
  SALES-ORD             &
  CHAR-DATE-EXPECT      &
  CHAR-DATE-SHIP        &
  CHAR-QTY-SHIP         &
  CHAR-UNIT-FACT        &
  CHAR-EXTENSION        &
  CONTRACT              &
  CHAR-AMT3

go
```

```
ACCOUNT=  SFD          GROUP=  DATA

FILENAME  CODE  ------LOGICAL RECORD-----------  ----SPACE----
                SIZE  TYP     EOF      LIMIT R/B  SECTORS #X MX

NEWLIN           122B  FA     35732     42000 185       16  1 31
ORDLINE   644    122B  FA         1     42000  65       16  1 31
```

FCOPY

To get the data into the new file we have made, we use FCOPY with the NOUSERLABELS option. This option prevents the file label from being over-written. If this happens, Powerhouse will no longer recognize the file as a subfile.

```
FCOPY from=NEWLINE;to=ORDLINE;nouserlabels

ACCOUNT=  SFD          GROUP=  DATA

FILENAME  CODE  ------LOGICAL RECORD-----------  ----SPACE----
                SIZE  TYP     EOF      LIMIT R/B  SECTORS #X MX

NEWLIN           122B  FA     35732     42000 185       16  1 31
ORDLINE   644    122B  FA     35732     42000  65       16  1 31
```

QTP on the HP3000

Below is the QTP routine to add the records in the subfile to the IMAGE dataset on the HP3000. Be sure that the process limit is set large enough to load all your records and that the target dataset capacity is sufficient.

```
run DATACONV
  request ORDLINE process limit 42000

<<Access your subfile and link it if necessary to get values for
fields that are not available in the subfile.>>
    access *QORDLINE link ("01" + OLD-ITEM) to ITEM of ITEMS  &
       link SALES-ORD to BUYER of ORDER-DETAIL

<<Define the numbers and their signs, so the sign can be properly
put.>>
define CHAR-DISCPCT1 char*6 = CHAR-DISCPCT[1:6]
define CHAR-SIGN      char*1 = CHAR-DISCPCT[7:1]
define NUM-DIS        num*10 = nconvert(CHAR-DISCPCT1) * -1 if &
        CHAR-SIGN = "-" else  nconvert(CHAR-DISCPCT1)
define CHAR-PRICE1    char*11= CHAR-PRICE[1:11]
define CHAR-SIGN1     char*1 = CHAR-PRICE[12:1]
define NUM-PRICE      num*10 = nconvert(CHAR-PRICE1) * -1 if &
       CHAR-SIGN1 = "-" else nconvert(CHAR-PRICE1)
define CHAR-QTYORD    char*11= CHAR-QTY-ORD[1:11]
define CHAR-SIGN2     char*1 = CHAR-QTYORD[12:1]
define NUM-QTYORD     num*10 = nconvert(CHAR-QTYORD) * -1 if &
       CHAR-SIGN2 = "-" else nconvert(CHAR-QTYORD)
define CHAR-QTYSHIP   char*11= CHAR-QTY-SHIP[1:11]
define CHAR-SIGN3     char*1 = CHAR-QTY-SHIP[12:1]
define NUM-QTYSHIP    num*10 = nconvert(CHAR-QTYSHIP) * -1 if &
       CHAR-SIGN3 = "-" else nconvert(CHAR-QTYSHIP)
define CHAR-UNIT      char*11= CHAR-UNIT-FACT[1:11]
define CHAR-SIGN4     char*1 = CHAR-UNIT-FACT[12:1]
define NUM-UNIT       num*10 = nconvert(CHAR-UNIT) * -1 if &
       CHAR-SIGN4 = "-" else nconvert(CHAR-UNIT)
define CHAR-EXT       char*11= CHAR-EXTENSION[1:11]
define CHAR-SIGN5     char*1 = CHAR-EXTENSION[12:1]
define NUM-EXT        num*10 = nconvert(CHAR-EXT) * -1 if &
       CHAR-SIGN5 = "-" else nconvert(CHAR-EXT)
define CHAR-AMT33     char*11= CHAR-AMT3[1:11]
define CHAR-SIGN6     char*1 = CHAR-AMT3[12:1]
define NUM-AMT3       num*10 = nconvert(CHAR-AMT3) * -1 if &
       CHAR-SIGN6 = "-" else nconvert(CHAR-AMT3)

    output ORDLINE     add on error report
      item DISCPCT             final NUM-DIS
      item COMP-ITEM-NO        final OLD-ITEM
      item COMP-CO             final COMPANY of QORDLINE
      item PRICE               final NUM-PRICE
      item PRICE-CD            final "CV"
      item QTY-BO              final 0
      item QTY-ORD             final NUM-QTYORD
      item SALES-ORD           final SALES-ORD of ORDER-DETAIL
      item STATUS              final "IR"
      item PRODCLAS            final PRODCLAS of ITEMS
      item COST-STD            final COST-STD of ITEMS
      item DATE-EXPECT         final nconvert(CHAR-DATE-EXPECT)
      item DATE-SHIP           final nconvert(CHAR-DATE-SHIP)
      item QTY-SHIP            final NUM-QTYSHIP
      item UNIT-STOCK          final "EA"
      item UNIT-FACTOR         final NUM-UNIT
      item PRICE-TYPE          final "CV"
      item EXTENSION           final NUM-EXT
      item ITEM-LOCN           final "    "
      item ALT-ITEM            final "               "
      item DISC-AVAIL          final 0
      item VENDOR              final VENDOR of ITEMS
      item AMT-FREIGHT         final 0
      item AMT-MISCHG          final 0
```

```
        item CONTRACT               final CONTRACT of QORDLINE
        item AMT1                   final 0
        item AMT2                   final 0
        item AMT3                   final NUM-AMT3
        item TAX-CD                 final TAX-CD of ITEMS
        item AMT-DISCMTD            final 0
        item WEIGHT                 final 0
        item CODE                   final "  "

go
```

Conclusion

Time Systems successfully converted from a PRIME computer to an HP3000 Series 948 running SFD in the allotted three months. We did not work eight hour days to do it, it was 12-14 hour days for everybody. But, we did it! Knowing some of these techniques ahead of time would have made our migration easier. I hope sharing our experience will make migration easier for you!

Paper # 3252

IBM S/36 To HP 3000/900 Migration

Jamie L. Harwood
Great Western Carpet Cushion Company
2060 N. Batavia Street
Orange, California, 92665
(714) 637-0110

So you've made the big decision to leave "Big Blue" and discover Hewlett-Packard. The next thing you know, you've got a delivery date, and you have to start your conversion. If you've ever done a conversion like this before, you'll know I'm not lying when I say that I never want to go through one again. It is a very long process, that without proper planning and TESTING, can be very painful. Hopefully this paper will help you to avoid some pitfalls.

Proper planning for the project is very important. About 3 months before we took delivery of our 922, (we now have a 948) I started to organize the existing menus and programs on our System 36. I used DOCUMINT (a documentation program) to generate listings of all the menus, procedures and programs in the system. Yes, it is a huge listing, but it is invaluable throughout the conversion process. It's amazing how all these menus and programs show up that you have forgotten about or just never knew existed! Some may even be useful again. Next I made up some simple forms to check off the menus and programs as we converted and tested them. These forms also come in handy when you are organizing your menus into accounts on the HP. Transferring the information is quite a time consuming process, but again, you do find some things you never knew about.

Next we started transferring our source code and procedures to 8 inch diskettes for conversion to DAT tape at our local HP office. This can be done using FROMLIBR to a disk file, then doing a SAVE to diskette. I did this alphabetically - it was the easiest way I could think of to keep track of it all. Being as the conversion was quite a long process, we had only 2 people working on it and it took about 9 months, you will probably have to keep track of any source modules and procedures that are updated or added to be brought over later, via a PC. (More on that later.)

The hardware conversion was planned next. Since the HP communicates to it's devices using RS-232, you can scrap all that twinax cable and use 4, 6, or 8 wire flat phone wire connected to a RS-232/RJxx connector. This very cost effective as well as easy to live with. This is a point to point connection, not a daisy chain connection like the IBM. Also, HP monitors and printers are not addressed, so they are incredibly easy to bring up. A performance notehere, the communication speed on the monitors can be increased to 19.2 kbs, even if the monitor configuration profile is set at 9600 bps. This can help screens come up that much faster.

Once we received our HP, we restored the source, load and procedures from the DAT tape that had been prepared at our local HP office. We then copied the source, load and procedures to their proper working accounts. The source and procedure conversion can also be done on your own using the conversion utilities provided by HP, CHANGER, PURGER, RST and OCLTRAN. If you don't have access to these on your system, GET THEM!! (It is called TRANSFORM UTILITIES by HP.) They are really helpful, and the only way to convert and compile your RPG programs and OCL procedures in volume. The provided utilities generate job streams to compile the RPG programs in compatibility mode, but a few changes can be made with CHANGER to compile these in native mode. OCLTRAN does a pretty good job of transforming the procedures, but some work does need to be done to them to make them completely workable. For example, the EVALUATE command is translated as EVALUATE.P. This will generate an error message when encountered and needs to be changed to just EVALUATE. The CHANGER utility in account CONV will do this just fine for ALL of your procedures at one time. Also, you will see lines in your procedures checking to see if a file is open, then waiting until the file is available. If you use a ";SHR;LOCK" on all of your file statements, there is no need to wait for the file to be available. Again, CHANGER can be used to insert the ";SHR;LOCK" into your file statements.

The Transform utility adds some file description specifications to your RPG programs the most likely will be unfamiliar - KLOCK and KNOLOCK. These Statements tell the program that either manual locking (KNOLOCK) or automatic locking (KLOCK) will be done within the program. File locking is required if you are using compatibility mode KSAM files. If you are using native mode KSAM files, a ";SHR;LOCK" on your file open statements in your procedure will invoke the transaction manager, and no other locking strategy is needed. As yet, I have only tested this with KSAM files, I do not have any experience with TURBOIMAGE files and the transaction manager. We happen to be using 99% native mode files, so we used CHANGER once again to comment out the KLOCK and KNOLOCK lines in our source modules, the used RST to recompile all of the source.

After the initial bulk of the programs and procedures were converted, we brought the remainder of them over via a PC. Using the same steps as before, we started with a FROMLIBR to a disk file on the IBM. Then, using a terminal emulation program on the PC, we brought he files from the IBM to the PC hard drive, converting them to ASCII files in the process. From there, we used ADVANCELINK on the PC to transfer the files to the HP.

We then used the Transform utilities to convert the programs and procedures. This sounds like an involved process, and it is, but it really doesn't take long, and you can do it any time you need. We brought over our data files using the same process. It works well, except for files with packed fields. On these files, we first wrote programs on the IBM to unpack the data in the files, then did the transfers. Once the files were on the HP, we wrote programs to re-pack the data. Yes, this is also an involved process, but again, you can do it any time on your own. Organization is very important here, so we had a checklist to check off each step of the process as it was done, as well as to keep track of the files we had converted.

Once the sauce programs and procedures are translated, the real fun begins. First off, not all of the programs compiled. The biggest reason was that the HP RPG compiler,in some respects, is a bit pickier than the IBM compiler. Double defined fields must be the same length, and decimal positions in numeric fields must be the same if the field is double defined. IBM RPG lets you get away with these bad habits, but not so on the HP! During testing you may also encounter a run time error, arithmetic overflow. Luckily the error message tells you the line where the error occurs, so it is easy to fix. At least it catches this before a report or a file is updated a digit short!

The last significant difference between the two versions of RPG is in the MOVEA operation. The MOVEA operation will only work with alphanumeric fields on the HP. So if you encounter an array the ends up blank, most of the time the culprit will be the MOVEA operation using a numeric field.

Screen entry programs, especially those with several overlapping screens will need some work. It's easy to spot these problems when you start testing - usually a screen will come up, look just fine, but none of the requested responses will work. For the most part, if the overlay screen is adjusted with a starting line number where the screen actually begins, (Form attributes in SIGEDIT) it will solve the problem.

At this point you will discover the lack of a field exit key on your keyboard. From a programming standpoint, this is somewhat easily fixed. But from a user standpoint, it takes a bit longer to get adjusted. From a programming standpoint, you have to design numeric fields appropriately in SIGEDIT. Usually, a numeric field can be defined with a zero or blank fill for the input. For output to the field, use a 'Z' edit code on the output specifications to eliminate the leading zeros. If the field may require a negative sign, it needs to be defined as a signed field in SIGEDIT. In this case you must also add one additional position to the field for the negative sign. Then, if you have SIGEDIT generate the input and output specifications, it will adjust the signed field accordingly for the sign to over punch the last digit in the file. If you don't define the field as signed, you will know it right away, as it will usually return a 9 in the sign position. When a numeric field is returned to the screen for modification, the change can be made to the field, then the remainder of the field must be cleared out by either hitting the clear line key or the spacebar. Some of our users will even use the clear display key first to clear out the entire screen. This was the biggest issue in the conversion with my users. We made sure to work individually with each of them before the conversion was made, but the first few weeks of actual usage on the the new system was a bit tough for them.

When you first bring up a screen entry program, you will notice that the function key blocks at the bottom of the screen will change their definition. To enable any of these functions, the entry fields in the Form Attributes definition in SIGEDIT must be left blank. To disable these functions, the entry field should be filled with an asterik. The opposite is true for defining command keys for the screen - the field is filled with an asterik to enable the key, and left blank to disable the key. Function key 2 represents the Print Screen key. The printout generated here will default to the system printer. Unlike the IBM, you cannot define default printers for your workstations in your system configuration, so you must direct your printouts using file statements with a "DEV=" in your procedures. To direct you print screens to a local printer, the statement is "FILE RSIPRINT;DEV=ldev #".

We are using PROCMON as a 'shell' to run our programs under. We have found that there are advantages and disadvantages to this. First off, PROCMON makes the system look the same as the IBM to the user. PROCMON also allows the use of a local data area (LDA) for the workstation, and PROCMON also establishes the RPG date for the programs - these are several very necessary items. Technically, though, there are some disadvantages to PROCMON. First off, PROCMON is a compatibility mode system, and therefore does not offer all of the 'neat stuff' available to you if you were simply using job streams. PROCMON also causes more overhead on the system, and this can cause some performance problems. There are also a couple of really important functions that don't work too well in PROCMON. One is RENAME - yes RENAME. I have found that if you use RENAME in PROCMON, you will get an end of file error message. We have gotten around this by using FCOPY in our procedures. But you have to be careful with FCOPY also. If you copy a native mode KSAM file several times using the new option, you will find that soon the file gets incredibly large and will eventually blow up. There are two ways to get around this - one, use a compatibility mode KSAM file, or two, build an empty file first, FCOPY into the file, reset the record pointer to 0 in the original file, then FCOPY back into it. The second solution sounds worse, except when you consider that file locking in your program is REQUIRED for compatibility mode KSAM files. So it comes down to the number of programs that have to be changed to accommodate the file locking.

Building new menus using BLDMENU in PROCMON is very easy. Although I would suggest using a regular text editor to modify an existing menu. We have had a few disasters trying to modify menus with BLDMENU, with the original version of the menu being destroyed.

All of these drawbacks are known at the HP Response Center, but they don't seem to be in a real hurry to fix them. I must admit, though, that the Response Center has been excellent in helping us with our problems, and helping us with work around solutions when actual fixes to PROCMON are not due in the near future.

A major issue about the HP system that came up a bit too late was the number of spooler printers supported in the system. The IBM had no limit to the number of spooled printers, so we had never considered that to be a concern. But, come to find out, it is! Our original 922 system only 'officially' supported 16 spooled printers. I had 27 set up as spooled. The system worked, but there were pauses in the system caused by all of those printers which severely affected the system performance. This was the biggest reason that we now have a 948, which supports 48 spooled printers. This may not seem like a big issue at first, but consider the fact that if you run a printer 'hot', only one user at a time can have access to that printer. Any subsequent user programs will wait until the printer is available. In most working situations this is not good. So please be sure to consider the spooled printer issue carefully in your conversion.

Aside from the limits on the number of spooled printers, there is not an HP system spooler interface. There are more than enough options available with the LISTSPF command, but it is quite a bit more cryptic than the IBM spooler program. We have developed several command files to make the commands easier for the users, and there are also some useful utilities provided by INTEREX. Also, if you are planning to incorporate non-HP printers into your configuration, the HP Response Center will help you to a certain point, then claim that the printers are not supported by HP and will close the call. Usually you can get enough help to get the situation solved, but once you cross that line - you are on your own. We have several Epson printers with serial interface boards in our system, and after some initial problems with environment files, they have been reliable as well as economical.

One of the most important reasons we chose the HP system was for the remote communications method. We have 2 remote sites on our system, and on the IBM it seemed like we were having problems quite often with remote controllers, as well as having to answer messages on the system each time a remote workstation or printer went down. Well, I have to say that in 4 months I have not had one single problem with the communications. It has been such a relief not to have to deal with the controllers and IPL'ing the system at the most inoportune times for a controller. For all of the work of the migration, this is one benefit that has made it all worth the effort!

In conclusion, the conversion was so much work, but we are finally seeing the benefits of it. Our users are comfortable with the system now, and the 948 has provided us with the performance we originally expected. We can also see the possibilities of a client/server system available to us, as well as some well proven 4th GL's. The Hewlett-Packard Response Center is extremely good, as well as friendly. I hope that some of the facts in this paper will help your conversion, because, believe it or not, it will be done some day. Good Luck !!

Paper 3253

Transporting an MPE Application to UNIX

Todd Saylor, Aaron Johnson
BI-TECH Software, Inc.
1072 Marauder, Suite A
Chico, CA 95926
(916) 891-5281

For our company the move to UNIX was more of a necessity than a choice. Like it or not, the market is moving rapidly in the direction of UNIX. As a software producer we would be foolish to ignore that fact. Just a few years ago we sold accounting software exclusively on HP3000 systems. Now we are finding only 25% of our sales interest is on that platform; the remaining prospects want a UNIX solution. Without such a system, we might find ourselves going the way of the Dodo bird or the Beta VCR.

There are reasons other than sheer momentum to consider a UNIX solution. While it is not a magical and perfect operating system, UNIX does offer several advantages over a proprietary system such as MPE. It is available on an increasing variety of hardware platforms. This removes many of the strings that typically tie a customer to a particular hardware vendor. If the need should arise it is conceivable, although not always trivial, to convert to another UNIX platform. A much greater variety of both hardware and software is afforded by the move to a standardized operating system.

Another advantage of UNIX is its programming environment. UNIX is by no means a "user-friendly" operating system; the average HP3000 user would feel quite lost on first introduction. After time, however, a programmer can feel very productive and even comfortable. Standard tools are available to perform nearly every simple function, such as printing the date, editing a file, or finding out who is currently logged on to the system. These tools can quickly be joined into powerful command files, known as scripts, which can be customized to accomplish nearly any task. Developing standards such as POSIX, XOPEN, and ANSI ensure, at a minimum, reasonable amounts of compatibility between the various versions of UNIX.

Where to Begin

Before jumping headfirst into a new platform, there are many issues which should be carefully considered. Is your organization permanently switching to UNIX, or adding UNIX support while continuing to maintain an HP3000 system? Will existing applications be migrated to the new machine or will they be rewritten from scratch? In our case, we recognized the need for a UNIX offering but were not prepared to abandon our HP3000 client base. We had neither the manpower nor the desire to rewrite our application and maintain two separate versions of our software. Our solution was to develop a set of tools which could take our HP3000 application and, with minimal effort, migrate the software to UNIX as often as necessary. Thus, we continue to develop and maintain the package on our

HP3000, while being able to offer a UNIX version with the same look, feel, and features as the original. This represents an incredible savings to us; there was no need to double our staff to support the new platform. Documentation did not need to be rewritten and trainers did not require retraining. Originally we had planned to release updates to our UNIX version six months behind the corresponding MPE release, but we are now able to send out enhancements on both platforms simultaneously. Eventually, we may choose to rewrite our system to create a "native" UNIX version. If we do so, however, we will have the comfort of successful UNIX version which is already in place, leaving us time to carefully construct a successor.

It must be decided whether to convert the application in-house or to purchase third-party migration tools. Migration tools can remarkably shorten the time it takes to complete a conversion. This can be especially important if you are trying to get a product to market quickly. Such tools are typically the result of much more research and development than a single company would put forward during a migration, so the solution would be significantly more complete and robust. When evaluating migration tools, compare the product cost to the person-years that would be required to solve the problem in-house, plus the opportunity cost of potential UNIX sales lost during the development period.

Other choices to be made include selection of a database platform, selection of a programming language or languages, and selection of hardware. HP3000 users typically move to an HP9000 UNIX platform, but other options should be considered as well before the final decision is cast. When choosing a machine, do not under-configure. Our experience has shown that multi-user applications do not perform as well as under MPE, given the same hardware capabilities. We attribute the difference partially to less evolved job scheduling methods. Other factors include the character-based terminal I/O and the work-intensive nature of relational database platforms. To resolve this issue, we always recommend a machine with more "horsepower" than the corresponding HP3000 solution would require.

It is important that these decisions be made and a plan be created before work begins. At best, a migration will take weeks of time and a lot of learning. At worst, several person-years could be spent developing a solution which is unworkable. Fortunately, there are steps you can take to greatly increase your chances for success. Begin by purchasing some general books on the UNIX operating system. We find that UNIX documentation rarely contains the level of detail or examples we would like, so it doesn't hurt to have several sources of reference. Develop a realistic plan for the migration and allocate enough resources to handle the task. Learn to use the many tools available under UNIX rather than struggling along without them. A little knowledge can make the difference between a productive experience and a frustrating ordeal.

File Transfers

Before a migration can truly begin, a method is needed to physically move files between the two machines. We have experimented with several methods; by far the quickest and most convenient is through the use of networking software. A network configuration such as Ethernet can be used with DSCOPY or "ftp" to quickly transfer files. The biggest drawback to this method is that none of the utilities we've used will support copying of filesets. To work around this limitation, we developed command files which will build a fileset on the HP3000, transfer the list of files to UNIX, then transfer each file in the list using repetitive commands within the DSCOPY subsystem. Using this method we are able to transfer all of our source code, approximately 600 programs and over a million lines of code, in 10-15 minutes.

For those of us not lucky enough to have the machines networked together, there are other, less elegant methods of file transfer available. There exists an unsupported HP3000 version of the UNIX tape archive utility, "tar". Using this program, it is possible to copy files onto media such as a DAT tape, then use the standard tar command to load the files onto the UNIX system.

Another method we've used is a serial connection between the two machines. This method limits the transfer rate to the baud rate of the serial ports, normally 19,200 baud. We are not aware of commercial software which will handle this task so we were forced to write our own file transfer software, consisting of a background process which runs on the HP3000 to service transfer requests and a "front end" program which runs on the UNIX machine.

The final method we employed was file transfer using a commercial telecommunications package such as Reflection or AdvanceLink. This required a two-step process; transfer HP3000 files first to a PC, then back up to the UNIX machine. We were able to set up command files under Reflection which would transfer entire filesets as one operation, but we were still limited by the size of the available PC disk storage. This is our least favored and slowest method.

Migration of Code

Source code is certainly the single largest segment of a migration. It seems reasonable to assume that an application written in a "standard" language will run equally well on any machine. This is not so; every implementation of a language is different. Most implementations, including HP3000 compilers, provide numerous extensions to the language. Many programmers find themselves using language extensions without even realizing their code is not portable. Our migration experience is currently limited to COBOL, Pascal, C, and SPL, but users of other languages face similar issues.

COBOL

Much of the HP3000 world uses COBOL as a primary development language. Unfortunately, the language's popularity is much weaker in the UNIX arena. This is not to say that the language is not available; there are a number of compilers to be found. However, we found compiler bugs were much more prevalent than in the MPE environment. Often we resorted to workarounds in order to keep our code running. Sometimes we found ourselves asking, "are we the only ones who use COBOL under UNIX?" On a more positive note, most of the bugs we found have since been fixed, although there is never a guarantee that they have been fixed on all platforms.

Both HP-UX COBOL and AIX VS COBOL are Micro-Focus implementations of the language, licensed respectively by HP and IBM. We chose them because they were readily available and fairly standardized. The two versions are nearly identical, although the defaults for several compiler options differ. Choose carefully when setting compiler options; the right set of options can save weeks of effort trying to code around missing features which could have been enabled. We were faced with a few trade-offs when setting options. For example, the OSVS option will support several desired features such as CURRENT-DATE and TIME-OF-DAY, yet setting the option disables other important features, such as the use of reference modification within expressions. In this case, we were forced to disable OSVS and provide certain features using our own external routines.

Once a compiler is selected, the real work can begin. Transfer a few programs over to the UNIX system and attempt to compile one or two. This serves the double purpose of teaching the compiler's operation and uncovering needed code changes. When compile errors are found, it is helpful to make corrections by hand, while making a list of all needed changes, until the program can be made to compile successfully. The resulting list is an excellent starting point for the code conversion process.

Unless the set of programs to be migrated is extremely small, some sort of conversion program is required. We created a process which takes either source code or COPY Library files, makes all necessary changes, and produces code which is ready to compile under UNIX. The resulting files need only be moved over to the UNIX system and compiled. This sounds quite simple, and it is, but it didn't evolve overnight. We migrated our entire application dozens of times, making changes to the converter each time, before finding this level of ease.

A useful code translator should perform many functions. HP COBOL extensions must be recognized and replaced with suitable equivalents. In many cases, MPE style filenames (FILE.GROUP.ACCOUNT) need to be converted to equivalent UNIX filenames (/account/group/file, for example). COPY Library modules should be processed during program translation, as they are referenced, so that all dependencies may be caught. Macros should be expanded and replaced in the source code, as the Micro-Focus compiler offers no macro support. Conditional compilation statements; for example, $IF X1=ON, must be evaluated and removed. Our approach was to set an MPE/XL environment variable which indicates the desired status of the compile switches, then analyze and convert the code as if those switches were set. The converter must strip all "$IF" statements from the code, so only one set of conditional options will be active in the converted version.

Once the resulting code is compiled, it must be linked with all necessary routines to form an executable module. UNIX does not offer the concept of an SL or an XL, so a new method of linkage to subprograms must be found. Using most languages, the only choice available is to compile each program into a separate executable, linking in all needed subprograms and system routines. Even if shared libraries can be used, the resulting program is larger than its MPE counterpart. To make matters worse, changing a subprogram may require that every main program be recompiled or re-linked. Fortunately we were able to make use of a few features of Micro-Focus COBOL to greatly reduce our grief.

The Micro-Focus compiler can produce four forms of object code: ".o", ".int", ".gnt", and true executable files. ".o" files are the UNIX equivalent of "OBJ" files; they contain executable code but must be linked in with other modules before being truly executable. The ".int" files are intermediate code, which is interpreted by the COBOL run-time routines. ".gnt" files are the "native" version of the code; they contain true executable code but must be invoked by the COBOL run-time environment (RTE), which is a separate program. We make use of most of these features by creating our own RTE. This is done by compiling all of our subprograms, including the MPE emulation routines, into ".o" files. These files are linked with the standard COBOL RTE routines to form a new RTE, with the ability to dynamically load main programs. Our main programs are compiled into ".gnt" code, which is invoked by our RTE. We named the RTE "run", so programs can be executed with familiar looking commands such as "run myprog". This way, changes to subprograms can be enacted simply by compiling the routine and creating a new RTE. Conceptually, the RTE is quite a bit like the XL or SL it replaces.

COBOL I/O will work correctly in the converted version, but will not be compatible with HP3000 features we tend to take for granted. File equates, file labels, and temporary files are all foreign concepts to UNIX. A UNIX file is simply a stream of bytes, with no fixed record size or file limit. This greatly complicates the issue if a program is relying on aspects of MPE files such as file codes or blocking factors. Things are even worse if special file types such as message files or KSAM files are used. We tried using standard UNIX files with linefeeds as record terminators,but we didn't find the consistency and reliability we needed until we started fully emulating the MPE file structures. Our file I/O routines now maintain file labels at the beginning of each file, providing vital information such as record size, file code, and user labels. We also support standard UNIX files to a limited extent, so files created with an editor such as "vi" can be accessed without problems. File equates are maintained and examined by the I/O routines. Temporary files are stored in a special "/tmp" directory which is unique to each session and is removed at logoff.

Our solutions work quite well on their own, as replacements for the MPE file intrinsics, but we are still left with the issue of COBOL I/O. Fortunately, Micro-Focus provides an interface which allows the standard COBOL I/O to be replaced by user-written routines. We just needed to write an I/O routine which accepts a predefined structure for I/O requests, performs the actual I/O as necessary, and returns an appropriate completion status. This gave us the control we needed, allowing us to use the power of FOPEN, FREAD, etc. wherever COBOL I/O is performed.

Another issue which must often be confronted is variable alignment. Under classic MPE, variable alignment occurs on 16-bit boundaries. The MPE/XL compiler uses 32-bit boundaries, but continues to support 16-bit alignment as an option. The UNIX platforms we use offer no such option; i.e., 32-bit alignment is required. Violation of this rule earns the message "bus error" as the program halts unexpectedly. The compiler does not warn about such problems; the programmer is left to discover them as run-time errors when incorrectly aligned variables are passed in procedure calls. To avoid this issue we recommend setting the 32-bit option on the HP3000 and correcting any problems there before proceeding with the migration.

Pascal

Our experience in migrating Pascal is limited to HP3000 to HP9000 conversions. The Pascal on those two machines is fairly similar, although some distinct differences remain. Porting to any other platform would first require finding a similar implementation of the language.

Under Pascal, all functions and procedures, including MPE intrinsics, must be declared before being used. Fortunately, the HP3000 intrinsic file mechanism is supported under HP-UX Pascal. We found it possible to directly move the SYSINTR file from the HP3000, then make any necessary changes to the file using the BUILDINT compiler directive. The only changes our intrinsic file required were due to slight parameter differences between some of our intrinsics and their MPE counterparts.

Many of the issues faced in a Pascal migration are the same as those previously discussed under the COBOL subheading. For example, MPE style filenames in statements such as INCLUDE must be converted to UNIX filenames. To this effect, HP-UX Pascal provides the +C option for automatic filename conversion. While this option is worthy of consideration, we chose to translate the names ourselves to gain more flexibility in directory naming.

As with most languages, Pascal I/O routines are probably the most difficult issue of code migration. Basic functionality is provided by the language, but true HP3000 emulation requires all I/O routines to be replaced. Otherwise the functionality of file equates, temporary files, and special files are lost. As it is not feasible to simply replace Pascal's I/O library, we chose to change the names of all I/O routines during code translation. For example, READ and READLN statements became x_read statements. We then wrote the new set of routines, employing the functionality of the MPE file intrinsics. As we did this, we broke down the large number of I/O procedures into more manageable groupings: APPEND, RESET, REWRITE, and OPEN were merged into a single x_open call with an additional "type" parameter. We also found it necessary to remove the INPUT and OUTPUT file declarations from the PROGRAM statement, and instead use our own I/O routines to open $STDIN and $STDLIST in the main program block. PARM and INFO declarations were also removed and replaced with our own function calls. UNIX offers a similar convention for parameter passing, but does not make the distinction between numeric PARM values and character INFO values.

Data alignment is another sizable issue. HP3000 code using 16-bit alignment will certainly run into problems when migrated. File layouts and IMAGE records often change size when alignment rules change, and the resulting errors are not always obvious. As in COBOL, we recommend using 32-bit alignment on the HP3000 system before starting the migration. Once found, most alignment problems can be resolved by simply changing variable definitions to packed or crunched data types.

HP Pascal real numbers come in two flavors: HP floating point and IEEE floating point. HP floating point is the "classic" MPE standard; IEEE floating point is the default on MPE/XL systems. Both are maintained as 64-bit values, but the latter is the form used on UNIX. It should be noted that any HP floating point data which is migrated will be meaningless unless converted.

Once the Pascal programs can be successfully compiled, some method of linking is needed. Initially, we found no technique which would approximate an XL or SL, so we were forced to statically link each program into a separate executable module. The modules were extremely large, so we put subprograms and MPE emulation routines into shared libraries and linked with shared libraries wherever possible. This reduced the executable to a manageable size.

C

Code written in C will be more portable than code written in other languages but there are still issues to be faced. HP extensions to the language, such as the long pointer type, will need to be removed. Dependencies on the MPE file system and MPE intrinsics run into the same problems experienced in the migration of other languages. Any program requiring a combination of C I/O routines and the MPE file system will need to be modified, much the way we modified Pascal I/O calls.

SPL

As one might assume, there is no equivalent language under UNIX. All SPL code requiring migration needs to be rewritten in another language. Pascal or C are typically used because of their similarities to SPL, but nearly any language may be used. We used a simple script, running under the editor QUAD, to automate basic SPL-to-C conversion tasks such as changing "BEGIN" to "{". For the most part, however, our conversion was manual and line by line. Other programs were rewritten in COBOL for better maintainability. Consider replacing all SPL code with another language on the HP3000 as well, as HP no longer sells a native SPL compiler.

Migration of Databases

Most of the HP3000 community uses IMAGE as its Data Base Management System (DBMS). Others such as ALLBASE are also employed, but IMAGE continues to be the standard. UNIX has no such standard so it is necessary to make a selection between the many database packages available. When moving to UNIX it is assumed that the database model will be relational, accessed through a Structured Query Language (SQL). Relational databases offer much greater flexibility than the network model used by IMAGE, but speed of data retrieval is characteristically slower. Put another way, IMAGE is designed to provide quick access to data by certain paths, while relational systems are designed to provide access to data by any path. In practice, many other factors such as buffering and optimization come into play, making the comparison less distinct.

Two of the most familiar names in relational database technology are Oracle and Informix. We have used both, although we currently offer only an Informix-based version of our software. Any brand of database should be usable for a migration, but some offer definite advantages over others. Some offer precompilers which allow SQL statements to be embedded directly in user programs, while others provide low-level routines which are called programmatically. Add-on features such as fourth generation languages, report writers, and form generators are often offered. Features should be examined carefully before the DBMS choice is made.

The first task in database migration is schema conversion. IMAGE schemas must be taken and translated into equivalent SQL statements for relational database creation. We chose to do this programmatically, although a one-time conversion might be done by hand. In the relational database world, datasets are referred to as tables, data items are called columns, and records are called rows. The basic idea is to convert all dataset definitions into SQL "CREATE TABLE" statements which define the type of every column in the table. For example, the following translation might take place:

```
IMAGE dataset definition               Informix table definition
NAME:       TEST-1, MANUAL;            CREATE TABLE TEST_1 (
ENTRY:      T1 (1),   [X2]             T1          CHAR(2) NOT NULL,
            T2,       [J]              T2          SMALLINT,
            T3,       [J2]             T3          INTEGER,
            XX;       [4X2]            XX01        CHAR(2),
                                       XX02        CHAR(2),
                                       XX03        CHAR(2),
                                       XX04        CHAR(2)    )
CAPACITY: 100;                         EXTENT SIZE 16  NEXT SIZE 16;
```

Most IMAGE data types map quite well into other databases, but a few require extra effort. The J4 data type, for example, is a 64-bit integer value which is rarely seen outside of COBOL. We were forced to map all J4 items to floating point types and convert the values at run-time. Data types such as the 'P' variety are assigned to SMALLINT or INTEGER columns, depending on the number of digits required, and converted at run-time. The concept of compound items does not exist on most RDBMS systems, so types such as 4J2 must either be broken down into multiple INTEGER columns or grouped together as a single, large CHAR field. Of course, this can lead to other problems because not all databases systems allow binary values to be stored within character fields. Even the names of items must be converted, as special characters such as the dash (-) are not available under SQL. Our conversion program changes all such characters to underscores (_). Fast access to data is required for all search items, so we use the "CREATE INDEX" command to generate indices on all search and sort items. This allows the RDBMS to access records quickly be key value. Search items are declared as "NOT NULL", informing the DBMS that the column may never be omitted.

The IMAGE "capacity" definition is roughly equivalent to extent sizing under Informix, a concept which closely resembles MPE/XL file extents. The biggest difference is that Informix disk space is assigned in large chunks which are typically shared by the entire database. Space for a single table is allocated one extent at a time, but there is no absolute extent limit. To assign chunks of space one can either allocate a disk partition, known as a "raw" device, or create a standard UNIX file, referred to as, of course, a "cooked" file. Performance is typically better with raw devices, as the UNIX file management routines are bypassed.

The toughest piece of database migration is certainly the run-time aspect. All IMAGE routines such as DBOPEN, DGET, and DBPUT must either be replaced or rewritten under UNIX. It is possible to convert each application program to use SQL statements rather than IMAGE calls, but we chose to leave the application software as-is and write our own versions of the IMAGE library routines. Simply put (no pun intended), we studied each IMAGE routine and wrote an emulation version which generates the equivalent SQL statement(s). For optimum performance, this was done at the lowest possible level. As far as the application can tell, it is still running under IMAGE. Any other methodology would probably end up being more of a rewrite than a conversion.

Emulation of DBGET requires an appropriate SELECT statement and one or more FETCH statements to actually retrieve the data. To explain the terminology, a SELECT statement requests a set of records. For example, 'SELECT T1, T2, T3 FROM TEST_1 WHERE T1 = "A"'. The requested set of records is associated with an identifier called a cursor. Records may then be retrieved from the active cursor using the FETCH statement. Generally speaking, serial and chained reads generate a select and multiple fetches, while all other modes of reading generate a single select/fetch combination. Some RDBMS platforms allow bi-directional reading of the selected records, while others permit only serial fetches. If the bi-directional feature is not available it will be necessary to create multiple SELECT statements whenever combinations of forward and backward reads (chained or serial) are needed. This can be quite complex because there is no simple way to start reading in the middle of a detail chain. Possible solutions to this problem involve maintaining the entire IMAGE chain structure, adding an additional sort field to every table, or disallowing this type of operation.

IMAGE classifies datasets into types: manual master, automatic master, and detail. Relational databases have only one type of table, so relationships between the tables must be maintained programmatically. Automatic master datasets could be eliminated because they are no longer needed as chain heads to access detail information. However, we chose to maintain them to more fully emulate the capabilities of IMAGE. For example, a serial read of an automatic master dataset may be used to list the contents of several linked detail datasets. The master/detail relationships must also be supported programmatically, or strange problems such as detail entries with no corresponding master entries may be experienced. To this end, our emulation routines perform many functions such as checking for master entries before putting detail entries, to ensure that database integrity is preserved. Naturally, our routines cannot do all this without knowing the original IMAGE structure of the database. To accomplish this, we maintain the relational equivalent of the IMAGE root file. Using this data we are able to fully emulate nearly every feature available under IMAGE.

While IMAGE sorts are applied as data is inserted, relational database systems sort data only on retrieval. Thus, all SELECT statements generated for sorted datasets must include an "ORDER BY" clause. Directed access may be accomplished using a "rowid", which is the relational equivalent of an IMAGE direct address. Unfortunately, not all RDBMS platforms use a 32-bit rowid. Some are quite a bit larger. Some are not consistent throughout the life of a row. Under Oracle, these constraints forced us to generate our own unique address and use it as an additional search item in each table.

Once these formidable issues have been tackled, all that remains is migration of data. In our case this was as simple as unloading the HP3000 data into flat files using a utility such as DB2DISK, transferring the files to UNIX, then writing our own version of DISK2DB under UNIX to load the data. These programs are quite simple; one uses DBGET to read data into flat files, the other uses DBPUT to load the files into the migrated database. If necessary, data conversion should be done at this point.

Migration of Screen Handlers

A large number of HP3000 applications make use of the V/PLUS screen handler. UNIX offers no direct equivalent, but a set of screen management routines known as "curses" is available. A big advantage to curses is its terminal independence. Using a terminal capability database called "terminfo", curses allows a single program to perform correctly on almost any brand of terminal. These capabilities can even be used in non-V/PLUS applications to allow, for example, inverse video capabilities regardless of the terminal being used. This has proven extremely useful, as each of our UNIX customers seems to have their own unique brand of terminal.

Many programs use "hard-coded" escape sequences to perform functions such as inverse video, drawing boxes, etc. These will continue to work under UNIX provided an HP terminal is used. However, converting the code to use curses should be considered to reduce dependence on a particular type of terminal.

Our approach to V/PLUS conversion was to programmatically convert each forms file into an equivalent driver file of our own design. The driver file is stored on the UNIX system and read in by our V/PLUS emulation software when the VOPENFORMF routine is called. The file contains a screen image and all necessary field definition and field processing information, including all editing information required to perform VFIELDEDITS. A compiled version of the driver file is maintained to improve processing speed; changed files are automatically recompiled upon first access. This allows our emulation routines, written under curses, to offer nearly all the functionality of V/PLUS and even add a few of our own ideas. For example, field editing happens on the fly; the simple act of typing data in a field causes the V/PLUS editing for that field to be performed. The emulation operates in character mode rather than block mode, allowing greater flexibility and control over screen editing.

Other Areas of Concern

We have covered the major areas of a migration, but many issues still remain. Not the least of these is the lack of MPE intrinsic library routines under UNIX. We solved this by writing our own version of each needed routine. This is easier said than done, but the final product is quite useful. We not support x% of all MPE intrinsics. With the exception of "option variable" intrinsic use, intrinsic calls within our application source code are often unchanged by the translation. The option variable routines, which have a variable number of arguments, needed only an additional parameter which acts as a bit mask, showing which parameters are actually present. This is the same method used internally by HP3000 compilers. Other slight changes are made to support features such as parameters passed by value, literal parameters surrounded by back slashes (\), and condition code (C-C or RETURN-CODE) values.

MPE file I/O is not easily emulated. Features such as file equates, temporary files, message files, and KSAM files are not standard on UNIX. As mentioned previously, we maintain our own file equates and file labels. We store temporary files in a special "/tmp" directory which is removed at logoff. Message files are implemented using message queues; a UNIX feature which is roughly equivalent to message files. We made use of the Informix product C-ISAM for our KSAM replacement. The functionality of C-ISAM is similar to KSAM, so the writing of our emulation routines was reasonably straightforward.

MPE users moving to UNIX might initially miss user defined commands (UDC's), but their functionality is easily replaced. Macros called "aliases" can be defined, or command files called scripts can be set up for more complex commands. The equivalent of a LOGONUDC is found in the .login and .cshrc files if the C Shell is used, or the .profile file if the Korn Shell is being used. For those not familiar with the terminology, a UNIX shell is the equivalent of the MPE command interpreter.

Other MPE features which are lacking under UNIX include the batch job facility and the spooler. UNIX commands can be run in the background simply by appending an ampersand (&) and jobs can be run at a lower priority through use of the "nice" command, but that is the extent of traditional batch job control. The UNIX machines we have used have no separate queues for batch jobs, no job priority, fence, or limit. We wanted more control over our batch jobs, so we created a background process, known as a "daemon", which manages a queue of batch jobs and responds to commands such as "showjob" through the use of a message queue. Batch jobs run under our own MPE shell, which is essentially a front end to our version of the COMMAND intrinsic. The MPE spooler was emulated using a similar daemon process. UNIX machines typically provide a spooling mechanism but do not provide features such as priority and outfence, so we wrote our own to regain the missing features. While these features are not strictly necessary for a UNIX migration, we wanted to lose no functionality in our software during the conversion process.

Conclusion

The ever-increasing popularity of UNIX makes a migration more and more attractive to most organizations. The task is never simple, but a little preparation can go a long way towards easing the process. It must be decided whether the migration is a one-time job or a continuing process. A hardware platform, database platform, and programming language or languages must be selected. Migration tools may be considered; when we began our migration we knew of none, but ours turned out so successful that we are now marketing them as a separate product, which affords the opportunity for a fully automated conversion. Whatever method you choose, UNIX compatibility is a worthwhile goal. A migration is always a challenge, but the finished product can be well worth the effort.

Paper 3255

The Realities of Mainframe Migration

Mark L. Symonds
Innovative Information Systems, Inc.
63 Nahatan Street
Norwood MA, 02062
(617) 769-7511

Over the past several years computing technologies have advanced to the point where we are able to realize significant productivity gains and cost savings. Since the start of business computing we have immersed ourselves in a centralized, monolithic mode of information processing. From the late 1950's to the early 1970's, this approach made sense and there really were no other viable alternatives.

In the mid 70's, businesses began to utilize midrange processors (ie: HP 3000, DEC PDP, IBM 3x) to accomplish processing for workgroups, departments, and whole businesses. The midrange computer made it economically feasible for a midsize business to purchase and operate a business computer system.

In the 1980's, the entire world's eyes were opened up to computers through the introduction of the microcomputer (PC). The PC made computers accessible to everyone from students to secretaries to entrepreneurs, and corporate executives. While the midrange made it possible for businesses to distribute computer processing into divisible units of work, the PC enhanced this concept and introduced us to the idea of open systems.

By combining these two leaps in technology, along with the help of distributed databases, networks, and development tools, business is now entering a new era of computing power. Increased power and decreased costs now place computers at all levels within an organization to meet competition in the national and global marketplaces. Companies utilizing corporate-wide computing capabilities can deliver what the marketplace wants first and service its customers fastest to reap the lion's share of rewards.

Making the move

Migrating from the centralized, mainframe mode of operations is not your typical conversion project. 'Right sizing' is not a simple process. It often means decentralized computing or moving to more than one computer system with a link to all computer users. But the end result puts processing where it belongs. Lead tracking system go to regional branches where sales people can follow up more quickly. Warehouse/inventory control system provide online control to manage inventory and fulfill customer orders more efficiently.

While a cost benefit analysis usually gains management commitment, the people issues of right sizing still demand attention. Understandably, MIS staffs may at first be reluctant to the right sizing concept. Existing systems represent considerable effort and commitment on the part of staff members, not to mention job security. Therefore, companies beginning to right size should educate all involved on the benefits of leaving the mainframe.

The People Priority

One of our operating principles is that PEOPLE ARE OUR GREATEST ASSET. Whether you are in the services, manufacturing, or distribution industries, our businesses are born and maintained by people. There are no machines or computers which can replace the creative energies of the human spirit. That is why I place the greatest amount of importance on our people and their involvement in the right sizing process.

Right sizing does not start with a migration project or systems implementation. It begins with the conception of the idea from the people within our organizations. Once the idea is brought to light, we become faced with a whole host of challenges. The most critical challenge is that of change.

Fear, that is what change represents to us. Most people want to squash any attempts at change. This is the nature of the human beast. Unfortunately, business CANNOT survive without change, and this is where the clash begins. In this scenario, many MIS people believe that mainframe alternatives represent job security. They immediately become defensive and try to eliminate any ideas about rightsizing. However, their insecurities, whether they are right or not, penalize the business.

In our experiences, education has proven to be the best medicine to assist in this paradigm shift. Businesses will continue to do what is necessary to survive, therefore, it is best to get on the train before it leaves. While many people will have negative feelings towards right sizing, we need to educate them so that they can participate in the change process. This provides several benefits.

First, many of our people have years of knowledge and insight in our organizations. They are probably best fit to assist in this technology shift. Secondly, education is a sign of commitment. It shows our people that the organization wants to invest in them, that the organization perceives value in their skills. Lastly, I have seen the layers of resistance peel away like the skin on an onion. However, this has only occurred when the people involved have been educated and understand where the business is going in right sizing.

One reason people fear right sizing is because workforce reduction is quickly assumed. It is true, we no longer require system programmers and large maintenance staffs. However, there will be many new roles introduced with client/server technology.

One critical role will be network administration. With distributed processing, interoperability, and scalability, we are seeing a proliferation of repetitive systems. When I refer to repetitive, I mean an application system that is replicated on many processors. As an example, an order processing system resides on small processors at each sales office. We do not want an operations staff at each sales office, all we need are the tools to manage the system from abroad. This is one new function for network administration. Along with managing remote systems comes the need to ensure that our networks are up and running 100% of the time. This requires highly talented individuals to work with end users.

We also need to address the shift away from maintaining software systems. In reviewing MIS budgets, we typically see 30% to 45% of the budget going towards maintenance. This is a very high number by midrange standards. Alot of this is due to the fact that the mainframe architecture has grown out of layers and layers of operating system software.

Today's midrange systems have been engineered with on-line transaction processing (OLTP) as the priority. It is very rare to find a transaction monitor on a midrange system, yet it is the lifeblood of the mainframe. Without all of the system software complexities, what do I do with my maintenance staff?

I can move my maintenance staff, along with the most of the maintenance budget, to system development. This is where MIS makes the philosophical shift from reactive maintenance to proactive development. We now have the opportunity to make a major impact on that large development backlog. And as I will discuss latter on, there are some great tools which will make us very productive and provide a high return of investment, and they exist today in our right sized environments.

We have reviewed how to work with our MIS staffs. There will be many changes to make, but it will definitely benefit our careers, as well as provide a healthy return to the business. But what about the end-user. These folks are also affected by this change.

Right sizing is providing many benefits to the end user. We are seeing new tools and graphical user interfaces (GUI's) which are providing great productivity benefits to the end user. GUI's to reduce the amount of training time required to utilize an application system. In industries with high employee turnover, this translates into great cost savings. GUI's also make systems less intimidating. Users no longer have to type in commands or navigate through cumbersome menu trees. GUI's allow users to relate to graphical objects. By concentrating on objects, it is easier to train users, and for users to retain system knowledge.

The steady shift of applications from mainframes to the client/server environment reflects the fact that the people who originate the information - at the desk, the department, and the regional office - often are in the position to make best use of it. Information in the field translates into faster customer response, increased customer satisfaction, more orders and increased profitability.

Businesses can now 're-engineer' themselves. Processing architectures will facilitate this process. Individuals have the power of their own cpu to work with, and the ability to share their data and knowledge with other members of their corporate community. The power of synergy can be realized. What I find most exciting is the capabilities that the user will now have. By allowing users to become more productive, by giving them more powerful tools to get their jobs done quicker and more efficiently, more creative thought time will be available.

This will result in a more competitive and productive business environment. Business can be more proactive to the climate, rather than reactive. This will also translate into the heart of the MIS organization because mainframes operations typically require more maintenance resources rather than development resources. Under the new technical environments, far less resources are expended on maintenance, which can translate into more resources for development.

As we have discussed above, the move to client/server architecture brings about alot of change, in the right direction. However, it is of the utmost importance that our people are treated as a high priority. The need for their skills and talent will be more important than ever because with this change, our organizations will be poised to become more proactive instead of reactive.

The Technical Details

Before we enter the world of client/server technology, I want to make one important statement. We are not advocating 'Shooting the Mainframe'. Mainframes still enjoy an important role in many organizations. However, there are considerable benefits in evaluating the offload or replacement of applications into the client/server world.

One technical detail we typically are faced with is that of drawing a one to one comparison of client/server technology with the mainframe. If this were possible, then client/server technology would not be client/server technology, it would be mainframe technology. Therefore, its time for another paradigm shift.

Lets review a couple of examples. First, there is the job control language (JCL). JCL is an important ingredient for any operation which utilizes batch processing, and I do not know anyone who does not use batch processing. However, many of the necessities of mainframe JCL are not necessary in the client/server world. I no longer need to allocate disc space or specify volume sets. Shell scripts in Unix are like a programming language. There is more power to be productive in these systems and less need for mundane details, which open operating systems handle automatically.

Another example is the need of a transaction processing monitor. There is no need for such a facility in the client/server world. The operating systems, databases, and many of the fourth generation languages account for these facilities. Unfortunately, this is a difficult concept for many mainframe programmers to grasp. These are just a few examples, but trust me, the technology is much different and much more advanced than the mainframe systems.

Distributing the Load

Distributed computing is the answer to growth in our corporate computer systems. For many of us, the need for increased processing power has meant upgrading and often replacing our current systems with larger processors. These upgrades often require $1 million or more. Distributed computing will allow us to put the processing and often times the information where it needs to be.

Order processing systems can move to the sales office, end user reporting can move to PC's, and compute intensive operations can be placed on hardware specifically designed for that function. In fact, the technology is getting so sophisticated that the operating system along with the network can determine which processor has availability and where is the best place for files to reside in the network.

All of this adds up to increased flexibility and growth while providing the appropriate level of resources for this business to operate efficiently and cost effectively.

Industry Standards

Because of the developments in system standards (ie: Open Systems) MIS and business can be alot more flexible in their choices of hardware, software and development tools. Standards has brought us the many 'abilities'. All of these abilities have helped to lower operating and development costs, as well as increase productivity.

With standards based hardware and software, the size of a system can be configured to match the organization it is supporting. This is called scalability. As an example, a company creates a sales force automation system. The system is developed in a 4GL. The 4GL can operate on any HP 9000. It also operates on a PC. Each regional office can have an HP 3000 of appropriate size (817, 837...). Corporate headquarters requires a HP 9000 Series 880/100. The field sales force uses laptop computers. All three groups (sales force, regional office, corporate headquarters) are utilizing the same software on the appropriate size computer. The portability of the software makes scalability possible.

In this example we see two types of cost saving. There is a one time investment in software development because the software is portable across platforms. The appropriate size computer is implemented with each business group due to the scalability of the architecture. What makes all the systems communicate properly is the interoperability of the systems.

The idea of scalability and portability is extended into the development environment. Many of today's development tools can be used on PC's, allowing a developer to be more productive. A developer no longer has to wait for a program to compile because the compile stream is competing against the production system. Many MIS departments are purchasing low end systems (HP 9000 S/817) for development. The software can then be ported to the production systems. This strategy provides a dedicated development environment whose test environment can mimic the production environment. Also, by being on a low-end cpu, the development license and support costs are less expensive than those on a larger cpu.

Development tools have also become more flexible and powerful. Under a NewWave or Windows environment, a program can multi-task his/her work. The programmer can be executing a program in one window, utilizing a debugger in another window, and reviewing the source code in a third window. And all of this is being performed on a dedicated cpu. Studies have shown that the Improved development tools along with a dedicated processor have increased development productivity by about 30%.

The increase in productivity does not stop at development. The implementation of client/server architectures have had a great impact upon business. Client/server systems place the power closer to the user, utilizing the features of scalability.

The net result of all of these benefits is a more competitive business. Downsized systems eliminate unnecessary costs, allowing the business to invest more resources in development or more money to the bottom-line (a favorite of many stockholders). Downsized systems can place the resources at the appropriate level within the organization, thereby allocating more computing power at less cost to those who need it. Systems can be developed more proactively and productively, allowing the users to become more proactive and productive in their functions. This has a rippling effect on the organization as a whole, making your company more profitable, productive and competitive.

Executing Change

We have now discussed many compelling reasons to choose right sizing. I felt it was important to review these issues because they are a critical component of the process. Remember, the process started with an idea by people which affects many people. As part of the education process, we must educate ourselves and our people as to all of the available options, and as we have seen, our options are endless.

A formula we have found successful in proving that right sizing is feasible is to select an application which can be migrated from the current architecture to our new platform. Part of the reason for this approach is to obtain an early 'win' in the right sizing process.

Right sizing is not a simple process. There are many challenges which lie ahead, technical, functional, and people. Moving from the mainframe is not a simple one to one exchange. The right size will likely require moving to more than one system. Remember, we are moving the processing to where it belongs, going from centralized to decentralized. This may mean putting a lead tracking system into regional sales offices, a warehouse/inventory system in each distribution center. This will impact data communications, the MIS staff, and the business users.

Once you have embraced a 'vision', you need to get down to the dirty details. First, you want to identify a candidate application. I would not recommend trying to migrate your universe. There is a learning curve involved, so a methodical, phased approach is important. What can be learned from the first application can certainly be applied on future project.

There are two basic approaches that can be taken in migrating a mainframe application. You can rewrite the application or migrate the current source code to the target hardware platform. Both approaches have their pluses and minuses. I will list them briefly.

REWRITE OPTION

Pros	Cons
Increase functionality	Retraining of MIS & Users
Improved maintainability	Lengthy development time
'State of the art'	Business risks
Best fit for new platform	Cost

AUTOMATED MIGRATION

Pros	Cons
'I know what I have'	status quo functionality
Length of project	(may not even be a con).
Ability to quickly prototype	
Reduce business risk	
Quick 'win'	
Leveraging an existing investment	

Performing a rewrite is very similar to a traditional application development project. Therefore, I will not even examine this option. The migration approach poses an interesting opportunity to due the brevity of the project and the ease of cost justification.

There are several products available today which allow a traditional mainframe user to migrate their applications from that environment to a midrange system. One tool we have worked with quite frequently is Conveyor, from Infosoft Gbmh of Germany. From this migration we have application programs up and running on the target system, 'bug for bug' (for lack of a better term).

Conveyor and other similar tools make the migration option very attractive. The ability to perform a prototype migration facilitates the learning curve and improves the opportunity of success. These tools do work, however they are not the total solution. The actual migration is only half the work. There are many other steps which I will present as part of our downsizing methodology.

Real World Examples

We are seeing strong evidence that mainframe migrations are working. We have witnessed many successes over the past five years. In the late 1980's, many were skeptical that a distributed approach was feasible. Today, many in the mainframe world are realizing that the move to Open Systems and Client/Server is not only reality, but that if they do not make the move soon, the competition will leave them in the dust.

Our organization has been part of several successful projects. Below, I will provide several brief examples of these projects and the benefits derived from them.

* A major pharmaceutical company migrated there order processing and bids and chargeback system from a mainframe, centralized operation to a midrange system. Batch processing was reduced from 4 hours to 20 minutes. The company saved close to $100,000 per month in operating costs. A fourth generation language was used to prototype and develop new systems. The project was completed within 6 months, and the functionality was increased to eliminate manual report gathering and allow EDI interfaces in order to create a closed-loop system.

* An international garment manufacturer migrated a mission critical system from a mainframe to an HP3000. This allowed for the system to distributed around the world on scalable machines. The result was a processor configured to support the distribution operation (cost effective approach) and a system which was created once and used in multiple locations.

* A major fine paper manufacturer had their financial application running on an outside timesharing system. The software was migrated to an in-house midrange system which had excess capacity. The migration project took under 8 weeks, and project provided a payback period of less than 3 months.

Getting From Here to There

Earlier, we discussed the planning process, obtaining management commitment, creating the vision. I believe this is necessary in any large-scale development project. These preceding steps are vital to any project of this nature. The following are a list of steps we have incorporated into our downsizing methodology. Detail is provided where necessary.

Experience has shown that the migration tools work. As stated earlier, downsizing is not a simple process. However, the rewards awaiting the project team, MIS, and business are immeasurable. Costs will go down, MIS will become more of a proactive development group and spend less time in a support role, and the business will benefit in terms of competitiveness, productivity, and profit.

Where can we Start

Downsizing may not be for everyone. There are alot of questions that should be asked before jumping into this process. Downsizing is a major commitment full of challenges and changes. Here are a few items which you may want to examine:

1. Compare your support costs for hardware and software against those of a comparable midrange computer.

2. Compare your software license fees against those of a comparable midrange computer.

3. Is over 30% of your programming staff dedicated to maintenance of current systems.

4. Compare the cost per MIP.

5. What is the cost of maintaining your current computer facility, including power usage, cooling units, cost per square foot? Is there a need to expand the physical facility?

6. How large is your development backlog and how productive is your current development backlog? Does compiling programs compete with production resources?

7. Are your systems mainly batch?

8. Would the business benefit from better functionality and improved response time?

These are some of the many questions you should be asking. The economic argument is easy to prove. The next step is to begin tackling the process. It is not simple (is anything in hi-tech simple?). However, the rewards are tremendous. Now that American business has entered the era of right sizing and fine tuning their operations, MIS has the opportunity to set the tone and lead the charge.

3256

Looking at UNIX with MPE Eyes
(The System Manager's Perspective)

Deborah Gray
Betsy Leight
Scott Trappe
OPERATIONS CONTROL SYSTEMS
560 San Antonio Road
Palo Alto, CA 94306
Phone: (415) 493-4122
FAX: (415) 493-3393

OVERVIEW

While any major transition from one operating system to another is vulnerable to slipped schedules and cost overruns, the transition from a solely MPE-based platform to the mix of MPE and UNIX holds unique surprises. This paper discusses problems related to system administration, so management can anticipate them, and adopt a proactive stance to provide a more cost effective and better planned transition.

PROBLEMS ADDRESSED

The conversion from a strictly MPE environment to a mixed MPE and UNIX environment presents some new challenges because of the radically different philosophical bases of each operating system. In addition, moving from the comfort and predictability of MPE to the unknown UNIX provides a variety of new technical delights, but it can also open a Pandora's box full of security and audit problems. While new state-of-the-art technology, attractive and easy to use graphical user interfaces, and robust networking become possible, the shock of a comparatively hostile operating system interface combined with accountability problems produces significant management challenges. This paper will present installation options for using the standard MPE and UNIX utilities and will offer suggestions for supplementing them with better software so that your shop will benefit from the decision to bring in UNIX.

System administration includes hardware and software configuration as well as ongoing maintenance tasks to provide data processing facilities to the user community. The administrator performs many functions in order to manage the multiple resources in the UNIX environment. These resources include CPU and peripheral hardware, user accounts, operating systems, and utility programs. Please note that we use the plural since most shops have more than one UNIX system. The administrator is faced with installing network hardware and software systems. Often, the computers are not of the same type, and a management nightmare is born.

Since the management tasks are many, the administrator needs programs and procedures that are easy to set up (install and configure), time efficient, easy to use, offer good performance, and result in high system availability. We also know that the administrator needs reliability so that tasks are repeatable and auditable by a third party. The MPE user is accustomed to getting hard copy audit trails of all system administration tasks. With UNIX, audit records are not always available.

SYSTEM ACCOUNTING

Management wants to know who is using what resources, how often, and when. In order to obtain this kind of information, a monitor process runs continuously in order to collect raw data such as user logins, programs executed, disk consumption, printer activity, and port connections. Once collected, the raw data is analyzed and organized into categories for management reports, user bills, graphical presentations, or data files. Very often, the summary information is exported to a master location and used for a corporate accounting application.

How is this done in an installation with both MPE and UNIX systems? With your MPE eyes, you see that system logging is accomplished by the monitor process. The MPE logfiles hold the raw data, some standard utilities (LOGUTIL) and command (REPORT) functions organize the data and generate reports for the system administrator. Third-party software takes this reporting to a higher level with price tables, consolidation options, discounts, minimum charges, and multiple-machine presentations.

Your UNIX system also collects system accounting information. The kernel monitors all tasks, the *wtmp* and *pacct* files store the raw data, and a suite of commands (*acctcon1*, *acctcon2*, *dskusg*, *runacct*, and many, many others) organize and report system accounting information. Only two price schedules (prime time and non-prime time) are standard with UNIX and few statistical reports are available.

The problem here is complexity. Your management wants more information for accounting and security purposes. First of all, we all want to know who is printing what, when, and how much. Then we want to have a complete picture of the installation. This means having system accounting utilities that capture and combine information for all machines in the organization. And with today's sophisticated graphical presentation tools, why settle for the same old reports when you can get a picture?

With your MPE eyes, you expect to see pricing tables, rate schedules, quantity discounts, minimum charges, and much more in the UNIX environment. This should be the minimum requirement.

WORK LOAD MANAGEMENT

Imagine you have 1700 UNIX boxes in separate locations all over North America. Your organization has an Order Processing application in place where some users enter orders, others send the orders throughout the network and, at some time during the day, the orders are summarized and acknowledged in a warehouse. This open systems environment has a client/server architecture and there is a great deal of work that must be done by the user community.

The system administrator cannot have one person at each of 1700 stations throughout the nation. He can use work load management software to schedule tasks, monitor activity, verify task completion and, if necessary, recover from unsuccessful operations. We expect this software will look for optimization opportunities as it monitors the realtime operation. What we hope for is an efficient software product that works in the background and maintains audit trails of data processing activity.

What the system administrator wants is a program or set of programs that automate launching jobs or tasks, optimize processing for maximum throughput, and record activities throughout the network. To get started, the system administrator must establish rules for each location including such items as task, date, requirements, and recovery.

With MPE, the user establishes rules with the STREAM facility. The environment is further refined and tuned with jobfences, priorities, queues, and job limits. The operating system dispatches jobs and the user can monitor activity with the SHOWJOB command. System logging generates an audit trail and users can perform manual STDLIST analysis to determine task completion. Automated recovery is not provided.

With UNIX, the system administrator uses an editor to define up to 26 processing queues and to set their characteristics. Users then establish their own job schedules with *crontab* for repetitive tasks, and *at* or *batch* for one time requests. The *cron* program runs continuously in the background and dispatches work, monitors progress, checks the exit status, and sends the task status and output to the task submitter. The user can monitor activity status with the *ps* command (process status). For audit purposes, the *pacct* (process accounting file) is updated whenever a task is complete.

Is this what you need in a multiple system environment? With your MPE eyes, do you see enough automation to get the job done? You need system administration utilities to automatically manage the workload and maximize throughput. You need a big picture view so you can see what is scheduled, what is running and what is completed on all systems in your network. Equally important is the need to have an easy method for defining workload and establishing processing rules. In fact, more rule granularity is required than standard UNIX supports. Scheduling rules include task, time, date, and prerequisites such as previous task checking, operator prompting, and output content analysis.

CODE MANAGEMENT

Operators and programmers often fight over the issue of code management. Who is responsible for it? The system administrator is usually pretty clear about two functional areas, program development and file distribution. Engineering is usually responsible for the former, Operations the latter. With a dozen UNIX boxes for engineering workstations, we have a problem that needs a solution. These engineers are all working on the same project, but each has his or her own assignments. During the program development phase, the engineers need some basic checkout/checkin procedures to avoid getting in each other's way. When the development is complete, the operators need an automated move-to-production process to handle file distribution.

System administrators solve the code management problems in several phases. Engineers identify library files and set up an official master index. Then, system administrators locate critical files, identify user classes, and restrict access to specific users. Next, they establish a process for basic checkout and checkin so that the engineers' work is preserved and the software asset is protected.

If the code management issues are related to production activities, then the system administrator defines an approval process and file distribution technique to take tested code from the engineers and move it to a production location. Standard separation of duties are implemented so that untested code cannot get in to the live production world.

Code management is enforced with proper implementation and audit trails are automatically generated for tracing purposes. With any code management implementation, release control and version control are necessary. In the production environment, automatic recovery and code rollback are vital.

What do we have for the MPE environment? Nothing comes standard with the operating system, but OCS/LIBRARIAN is a third-party industry standard. It handles the checkout/checkin and sophisticated move-to-production procedures.

In the UNIX world, engineers have two standard packages, SCCS and RCS, for source code management (notice the word "source"). With SCCS, the *admin* function is used to set up a library. Engineers access files with the *get* and *delta* commands. Audit trails are maintained with the *prs* function. If the engineers prefer RCS, the *rcs* command is used to define the master library. Checkout and checkin functions are executed with the *co* and *ci* commands. The audit trail is maintained with the *rlog* command.

SCCS and RCS are entrenched in the UNIX world. But, with our MPE eyes, we demand a more powerful, comprehensive, multi-system solution. First of all, the administrative setup must be easy to use. We must be able to associate more than one file to a class category and then use that category to access a collection of related files. Enforcing procedures must be transparent to the engineer because, as we all know, they have better things to do than spend time learning new commands in order to gain access to their source code (and usually resent the access controls in the first place!). In addition, generating audit trails must be automatic and comprehensive.

Operations management needs an automatic method to distribute files across the network. Automation ensures that object code is matched to source code and that the tested version of object code is sent to the correct production platform. After all the hard work that engineers accomplish, it would be criminal to compromise on sloppy (or non-existent) procedures for moving code into a live environment.

CONCLUSION

Recognizing the significant differences between MPE and UNIX is a good first step to planning the System Administrative procedures for your environment. The demand for robust and efficient system utilities is obvious to the new user since the standard programs do not handle multiple machines and do not automate repetitive tasks. The utilities that do exist are difficult to use and have limited online help facilities. When the EDP audit is performed, the administrator will have a hard time finding comprehensive audit trails and evidence that disaster recovery procedures are available.

Don't settle for less than you need! You will pay for it twice – once when you identify the missing utility, backtrack, and locate whatever evidence may exist, and a second time when you write your own comprehensive utility program. How much pain are you and your users willing to endure? Who do you want to pay? How much are you willing to pay? When do you want to pay?

Be strong and take action before this environment eats you and your users alive. It is not smart to grin and bear it; you are advised to either write your own system administrative utilities or purchase third-party software. The decision is yours after you estimate the costs of make versus buy. Remember, you deserve the best!

Meeting the Network Challenge: Running Multiple Network Protocols on a PC

Randy Robinson
Walker Richer & Quinn, Inc.

The development of the Network Driver Interface Specification (NDIS) by Microsoft, and the Open Datalink Interface by Novell have allowed network software vendors to develop multiprotocol products for the PC. These products bring enhanced capabilities to the PC user such as the ability to simultaneously access multiple hosts, LAN servers and other PCs on the network. They also bring increased responsibility to the network manager in terms of configuring each user's PC for access to the network and managing end-user access to all nodes on the network. Every station on a Novell or LAN Manager network can now also be a node on the corporate network.

Network Driver Interface Specification (NDIS)

The Network Driver Interface Specification was created by Microsoft and 3Com for LAN Manager. It has become a popular standard because it solves two key networking issues:

1) It defines a common interface to a network driver, enabling a single driver to support many different protocols (device independence);

2) It provides concurrent access to a network adapter by more than one protocol (multiprotocol capability).

NDIS drivers exist for most Ethernet boards, including boards from Novell, Western Digital, Hewlett-Packard, Racal-Interlan, Ungermann-Bass and 3Com. NDIS is used by Microsoft LAN Manager, HP's ARPA Services and Network Services, WRQ's Reflection Network Series, and many others. Though most commonly used in Ethernet networks, NDIS drivers also exist for Token Ring boards, including IBM and 3Com boards.

The higher layers in an NDIS configuration are called protocol stacks. Though a protocol stack may include separate programs for TELNET, FTP and TCP/IP, only the lowest of these layers (in this case, TCP/IP) talks to the NDIS driver. The higher layers (TELNET and FTP) then talk to TCP/IP.

An NDIS setup includes several standard components for the lower layers, and a choice of compatible protocol stacks for the higher layers. The lower layer pieces are:

1) PROTMAN.DOS - the NDIS protocol manager. This module serves as the coordinator for all the modules participating in the NDIS suite, and is loaded with a DEVICE= line in CONFIG.SYS.

2) PROTOCOL.INI - the protocol initialization file. This ASCII file provides configuration information for device drivers and binding information for the higher layers. The format is similar to that of the WIN.INI and SYSTEM.INI files used by Microsoft Windows.

3) An NDIS compliant device driver, e.g. ELNK.DOS for 3Com's 3C501 Etherlink board. Loaded in CONFIG.SYS, this driver will support any higher layer protocols that bind to it woth BINDINGS= statements in PROTOCOL.INI.

4) NETBIND.EXE - the network binding utility. Executing NETBIND (usually from within AUTOEXEC.BAT) initiates the binding of protocol stacks to the device drivers. If a single stack will be using the device driver, NETBIND will bind the protocol to the driver and exit normally. If more than one stack wishes to use the driver, then NETBIND will TSR and act as a switcher, coordinating the multiple protocols.

Open Data-link Interface (ODI)

The Open Data-link Interface was developed by Novell for Netware. Although this specification was announced some years ago, it has only been recently that the final specification was released by Novell. For that reason, NDIS implementations are more prevalent than ODI implementations. ODI solves the same issues of device independance and multiprotocol capability that NDIS solves.

Currently, there are fewer ODI drivers available than NDIS drivers, but that is changing.

As with NDIS, the higher layers in an ODI configuration are called protocol stacks, and only the lowest layer in the hierarchy talks to the ODI driver.

The lower layer components of the ODI setup are:

1) NET.CFG - the network configuration file. This is an ASCII file that has information for device drivers and the higher layers. It's function is the same as the PROTOCOL.INI file in the NDIS setup. (Unfortunately, it's format is not the same as the PROTOCOL.INI file.)

2) An ODI compliant device driver, e.g. 3C501.COM for 3Com's 3C501 board. This driver is a TSR rather than a DOS device driver, and can be uninstalled when it is not used.

3) LSL.COM - the protocol manager. This module manages the network traffic, switching packets between protocols. This module is also a TSR, and can be uninstalled when not in use.

NDIS vs. ODI

The PC network operating system, the availablity of compliant board drivers, and the architecture of the protocol stacks all determine whether the PC is configured with NDIS or ODI.

Most network software supports the NDIS standard. Many vendors even supply NDIS compliant IPX drivers. This allows Novell's Netware to run in an NDIS setup.

Fewer vendors support the ODI standard. Novell supplies an NDIS to ODI convertor that allows NDIS compliant protocol stacks to run in an ODI environment.

A good rule of thumb for deciding whether to use NDIS or ODI is: The fewer pieces, the better. It is preferable to use NDIS in a LAN Manager environment, and ODI in a Netware environment. Unfortunately, ODI compliant network software is generally less available than NDIS compliant software.

Virtual Terminal Protocols

PC users generally want access to their file servers, e.g.LAN Manager and Netware, and to their host computers, e.g. HP 3000s and HP 9000s. The file servers use protocols such as NetBEUI, IPX, and TCP/IP. HP 3000 computers can use either NS/VT or TELNET as a virtual terminal protocol. HP 9000 computers use TELNET as a virtual terminal protocol. Both TELNET and NS/VT run on top of TCP/IP.

TCP/IP Configuration

Configuring a PC to use the TCP/IP protocol can be very confusing to the uninitiated. Familiarity with a few basic terms is necessary to ensure proper configuration.

1) IP address - all nodes on a TCP/IP network must have a unique address. This address is a32-bit number, consisting of four eight-bit parts, such as 192.146.38.201. Each PC on the network must have its own IP address.

An IP address has a network portion and a node portion. All the nodes on the network must have the same network portion. For example, the network part of the IP address can be the first of the four parts:

```
              192    .XXX.XXX.XXX
             /           \
        Network portion       Node portion
```

A full discussion of IP addresses would involve a description of IP address classes and subnetting. This is more complex than the scope of this paper.

2) Node name - nodes on a TCP/IP network also have three part names, such as MYPC.COMPANY.COM. These names are used when connecting to an HP3000 using the NSVT protocol. (They may also be used by the TELNET protocol if the network has a Domain Name Server - see below.) The three parts of the name are NAME.DOMAIN.ORGANIZATION. It is a good strategy to have all the nodes on the network use the same DOMAIN.ORGANIZATION. Each name must be unique.

3) Hosts file - host files are used when connecting to hosts that use the TELNET protocol, such as HP 9000s. The hosts file contains a list of host names and IP address. TELNET will use the hosts file to resolve the names of hosts to which users connect. For example, an entry in a host file may be:

```
192.146.38.101  myhost MYHOST
```

When a user requests a TELNET connection to 'myhost', the TELNET software looks in the hosts file and sees that 'myhost' has the IP address 102.146.38.101, and is then able to make a network connection to that address. The names is hosts files are case sensitive. That is why there are usually two name entries for each IP address. This allows the user to type the host name in either upper or lower case.

Hosts files are easy to construct. Without a hosts file, users would have to remember the IP addresses of all the hosts to which they connect. It is much easier for users to remember names than strings of numbers.

4) Domain Name server - Domain Name Servers can take the place of hosts files for TELNET connections. A Domain Name Server is a process that runs on a node on the network, and is able to resolve a host name to an IP address. A PC can be configured to use both a Domain Name Server and a hosts file.

5) Gateway address - gateways link remote networks to local networks. Many small networks do not have a gateway. Large networks usually do have a gateway. If there is a gateway onn the network, configuring the gateway address will allow users to connect to remote hosts. If there is a gateway on the network and the address is not configured on the PC, then the user will still be able to connect to local hosts.

Putting it all together

To build a working system, you need to start with a network adapter that is supported by an NDIS or ODI driver. From this base, you can configure a multiprotocol environment that provides access to file servers and hosts.

PCs with multiprotocol capability allow the network manager to dispense with RS-232 connection to host computers, and to get maximum use from the local area network.

CLIENT/SERVER COMPUTING: TAKING THE FIRST STEP

Doug Walker, Vice President of Development
Walker Richer & Quinn, Inc.

Has the age of client/server computing arrived? Judging from the number of articles in industry publications, an observer might well draw that conclusion. But what does the term "client/server" really mean, and what is its impact today?

This is a time of difficult choices. MIS managers find their budgets routinely slashed, while at the same time they are charged with upgrading their systems. What is the best approach? Should you scrap your current systems, or wait to see what the new technology provides? Or is there a middle ground?

I believe there are alternatives—solutions that let you prepare for tomorrow without sacrificing present performance. But before we discuss those options, we need to define "client/server" and discuss the context in which we use it now.

The client/server model

There is no one definition for the term "client/server." In this paper, client/server refers specifically to a distributed computing model in which presentation services and processing are split between a PC workstation and one or more larger host computers. The software is specialized on both ends: the "client" is the application making the request, and the "server" is the application providing the response. Though the client often resides on the PC and the server on the host, where each resides is not important.

When industry experts paint a picture of an ideal client/server model, they depict a world in which a user seeking information simply sits down at the PC, enters a few keystrokes, and receives the requested information transparently from the source or sources on which it resides—mainframes, minicomputers, or other PCs. The server application determines which records satisfy the requests and performs some preliminary processing before sending them to the client—it is not necessary to send complete files or records.

That world has not yet arrived. A number of developments must take place before client/server applications are developed that work across multiple platforms, networks, and systems, each with its own protocols and proprietary specifications. However, there are client/server applications that exist today. While most still use the PC primarily for presentation services and keep the bulk of the processing on the host, they are important first steps toward that ultimate goal.

The advantages of client/server computing

Why client/server computing? The client/server model is very attractive because it allows companies to combine the strengths of both the PC and large computers. PCs offer superior presentation services through GUIs (graphical user interfaces). Putting presentation services on the PC frees the host for more efficient use. PCs also can provide substantially more processing power on a MIPS per dollar basis.

The PC is an even more compelling resource because Microsoft Windows, and other such PC operating systems provide an attractive GUI, memory management, multiprogramming, and interprogram communication. Large computers, on the other hand, offer security, reliability, and unmatched storage capacity. Also, most applications that are used in the day-to-day management and control of a company have been written for, and reside on, host computers. Using both of these resources together greatly expands their capabilities.

Problems: the company viewpoint

Although client/server implementations seem to promise so much, they also present great difficulties. Companies have several issues to consider:

•Organizations have invested enormous amounts of money, time, and personnel in their current systems. Even if significant improvements could be achieved, how many companies could afford to dump their HP 3000s and adopt new technologies at once? What makes the idea even more untenable is that most HP 3000 systems are performing very well. And those systems are critical to day-to-day operations.

•Because the HP 3000 has such an important role, organizations are understandably reluctant to make changes requiring a wholesale shift.Customized client/server applications for the HP 3000 environment are available, but they are not easily put in place, demanding extensive knowledge of both PC and host programming. Even the specialists do not find client/server applications easy to write. What is even more problematic is that moving an application to a client/server model requires a fundamental restructuring—a risky procedure at best.

•The prospect of replacing existing systems with UNIX-based hosts using relational databases is also more than a little frightening. The conversion process carries the potential for disaster, presenting porting problems and the difficulties of dealing with a new development language, as well as tremendous expense, involving new hardware, new software, additional training, and perhaps new personnel.

•PC LAN-based systems do not provide any real alternative. Although they can be developed parallel to an HP system, they are unlikely to offer a solution for the near future. Security provisions are not as well developed as in the minicomputer and mainframe world, and the mean time between failures is much lower for LAN-based systems. The technology is simply immature and needs more time before it can be considered for mission-critical applications.

Problems: the industry viewpoint

Full-fledged client/server applications will not be developed until hardware and software vendors adopt standards and specifications for a variety of interfaces that will allow interprocess communications between applications and host databases and host servers, independent of protocols and systems.

The long-promised OSI networking standard, for example, is still some time away. In spite of all the talk of "open systems," there is no universal standard. While TCP/IP is the *de facto* standard by virtue of its wide acceptance, it has no formal sanction. And even with TCP/IP, there is still no standard for the virtual terminal interface. The HP 3000, for instance, requires a non-standard version of Telnet, NS/VT, for its block mode applications.

In addition, there are no set standards for interfacing applications to transports—an important element that must take place before client/server computing can enter its golden age. We need to move beyond proprietary socket interfaces such as NetIPC, and adopt standards for multivendor networking. Even Berkeley Sockets, which has sometimes been regarded as a standard socket interface for TCP/IP, has a number of different implementations—so much remains to be done in this area.

Multi-database server synchronization, and mechanisms to address specialized printing and data services (such as images, text, and mail) also must be addressed. As with all developments before it, the client/server model is an evolutionary process.

In spite of the difficulties, considerable progress is being made. In addition, there are interim solutions systems managers can adopt to position their companies to take advantage of these new developments.

Interim solutions

Today the industry appears to be moving toward full interconnectivity, allowing the interchange of files and data across platforms and network environments. Yet this effort becomes possible only when vendors support a common standard or specification. For example, TCP/IP has emerged as the standard for networking. Another is the NDIS (network driver interface specification), which allows network protocol stacks from different vendors to share a single network interface card, thus allowing a single PC to run several protocols. By allowing an almost transparent integration of different hosts and LANs, this standard promotes multivendor connectivity and makes it easier to distribute applications.

Developments are also proceeding in other areas, led in large part in the PC arena by the popularity of the Microsoft Windows platform.

Client/server development and MS-Windows

Microsoft Windows is a logical partner in the development of client/server-style applications. Windows provides memory-management to boost computing power and a superior GUI, and because it is such a popular platform, its specifications are quickly becoming industry standards. In addition, Microsoft is working with independent software vendors and others to develop standardized sets of both front-end interfaces (application program interfaces, or APIs) and back-end interfaces (service provider interfaces, or SPIs) to enable a mix and match of client and server applications. Microsoft describes this movement toward shared standards as the Windows Open Systems Architecture (WOSA).

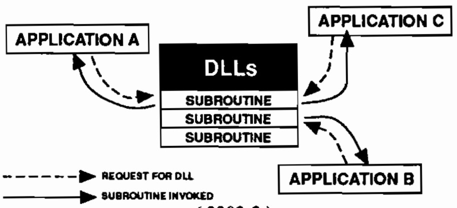
The idea behind WOSA is that developers will be able to write applications that are independent of networks, printers, mail, and file systems. If applications do not have to contain code for a number of different network or printing drivers, or address a variety of different protocols, for instance, developers can write them more quickly. They will take up less memory on the user's machine and—most important—they will pave the way for full interoperability and the ideal client/server world.

Even today, the Windows environment provides a number of benefits. Applications that use the Windows GUI can take advantage of icons, colors, various fonts, and bitmaps to present information to the user. With buttons, pull-down menus, and "drag and drop" function

operations become somewhat intuitive, allowing even those with minimal computer literacy to quickly become productive. In addition, several programs can be active simultaneously.

Microsoft standards already allow a much higher degree of multivendor compatibility than we have enjoyed in the past. For example, many vendors are taking advantage of Microsoft's standards for DLLs (dynamic link libraries). DLLs are subroutines that are invoked only when needed, and they can be shared by several applications. Because DLLs can be used again and again, a developer need not write new code to accomplish the same task with each succeeding application. In addition, with a published standard, different vendors can easily support the same functionality.

DYNAMIC LINK LIBRARIES (DLLs)



In other areas, DDE (dynamic data exchange) and OLE (object linking and embedding) provide standards that allow applications to exchange information. Winsock, a standard interface linking Windows applications to transports, is a possible solution to the problem of proprietary socket interfaces. Although such a "modular" approach to application development is in its infancy, in a relatively short time it should be possible for developers to use a number of standard methodologies and interface design tools. Linking applications with other applications or programming languages, or with transport protocols, will become much easier. These are the kinds of developments that are needed to speed the time to market of all applications, but particularly for client/server applications, which require a great many interconnections.

Solutions that use Microsoft Windows

The Microsoft Windows environment presents some attractive options for systems administrators who do not want to abandon smoothly functioning systems, but who wish to take advantage of some benefits of the client/server model.

The first step in client/server computing is designing a more attractive user interface. Terminal-based interfaces are the norm in the HP world, but users, now familiar with GUIs, desire them for host-based programs as well. As PCs have opened computer use to those who are less technically oriented, ease of use has become more important.

Designing a new user interface is possible with Microsoft Windows and other windowing technologies. Many tools have been introduced that allow the development of integrated front-ends that take advantage of the Windows GUI. Borland's Object Vision, Asymetrix's Toolbox, and Microsoft's Visual Basic are examples.

The Cognos approach

Other options for providing a new front-end include products such as PowerHouse Windows from Cognos, an extension of the PowerHouse 4GL application development environment. With PowerHouse Windows, a developer can design an application that presents a GUI running on a PC under Windows that communicates with an application engine running on a host computer.

Basically, a PowerHouse Windows application consists of a series of screens that are connected hierarchically and that together implement the functions of the application. Each screen describes the user interface and the data and processing necessary to provide one portion of an application's functionality. The user interface information is distributed to the PC (client) while the application processing runs on the host (server).

The user interface is described with a form design that extends the information contained in each screen design. There is essentially one form for each screen in the application, although the screens can also share information. Multiple screen threads allow an application to have more than one screen active at a time. Designing the user interface is simple with a WYSIWYG interface for forms design.

The Cognos approach offers some distinct advantages: it provides an easy way to provide an intuitive user interface for an existing application. It does not affect the processing of the basic application, and it builds on the growing Windows environment.

There are those who will say, "So what?—building a new user interface is only a cosmetic change." Since the processing remains on the host, the application is not truly distributed. And they are right. But although the industry publications may portray a world of unhindered, free information exchange across environments, platforms, and a multiplicity of networks, the fact is that today designing a new user interface is the "cutting edge" for many companies. They need not apologize for that.

The company in which I am a partner, Walker Richer & Quinn (WRQ), is known for solid terminal emulation and networking software. We often speak with customers in the field to determine which features they like in a particular product. Many times, we are surprised to hear that what they like most in our product is not necessarily the more advanced functions, but rather something we consider more basic, such as keyboard mapping. If you talk to our own developers, they will certainly agree keyboard mapping is important, but it is probably not the first thing they will mention about the product. And yet, time and again, thecustomer says, "Because of keyboard mapping, I can take a person who knows nothing about the HP 3000 or an HP terminal, and in a very short time, have them working on host applications." Those of us in development tend to underestimate the tremendous productivity gains that result from simplifying operations for the user.

Designing a new front-end does not provide full-fledged client/server computing. But it is a move that companies can easily make without jeopardizing existing systems.

PPL for Windows

At WRQ, we are also working on client/server enabling tools that take advantage of Windows technology and a modular approach to network transport interfaces. Our Process-to-Process Link (PPL) for Windows product, the first component of our Reflection for Windows Software Developers Kit, allows companies to move to client/server computing at their own pace.

PPL for Windows provides developers of HP 3000 applications with:

- Message Exchange
  Allows interprocess communications between Windows applications on the PC and an application running on the host regardless of underlying transport protocols

- The HP PPL Toolkit
  Gives a Windows application the ability to execute over 80HP 3000 MPE system intrinsics

- Fast File Transfer
  Delivers very fast file transfer for Windows PPL applications

Many DLLs are included. The PPL for Windows API is supplied via a Windows DLL, so any Windows development environment that supports calls to a DLL can be used to develop Windows PPL applications (including C, C++, Visual Basic, and others). The execution of host intrinsics from a Windows application is also accomplished through DLLs.

PPL shortens development time, since developers do not have to deal with constructing the links to transport protocols, nor writing the host portion of a client/server-style application. Once they have an operational client/server-style application, they may wish to go further—but they are not required to.

PPL for Windows is neither a fully-developed client, nor a complete server; it is an enabling tool. But with links to transports and the advanced functionality provided with DLLs, PPL allows companies to take the first steps toward client/server computing at their own pace. Using PPL, programmers can develop a new user interface, or they can go on to create applications that distribute more of the actual processing.

Neither applications developed with PPL nor with Cognos require a networked environment. Of course, in the ideal client/server world, networking will be essential, providing almost unlimited access to information. The point to note is that the first step can be taken even before a network is in place.

Conclusion

True client/server computing is still some time away. Developing client/server applications is very hard work, requiring in-depth knowledge of both host and PC programming. For most companies with established, smoothly functioning HP 3000 systems, switching overnight to a new technology is risky and impractical. But there is a middle ground that allows MIS managers to take advantage of many of the benefits of client/server without great risk or expense. That way lies in using the programming tools that exist today, anchored by widely-adopted standards and existing GUI technology.

Most companies are looking for ways to make optimal use of their resources. They are looking to software and hardware firms to provide technologies that will serve as interim solutions that can be upgraded in time. And, by building on an established and growing platform such as Microsoft Windows, and *de facto* networking standards such as TCP/IP, organizations are not likely to find their efforts quickly outmoded.

Solutions do exist. They are real. They are practical. If all an organization wants to do is provide a more attractive front-end for a host-based application, let us not dismiss that. Let us say instead, "It is the first step toward client/server computing." The rest will surely follow.

PAPER NO.: 3304

TITLE: Client/Server Technologies in the 90's

AUTHOR: David Haberman
Speedware Corporation
1150 Hancock Street
Suite 410
Quincy, MA 02169
(617) 774-0440

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

PAPER NO.: 3306

TITLE: What Does A Distributed System Give You?

AUTHOR: Niket Patwardhan
Oracle Corporation
400 Oracle Parkway
Mail Stop 659 411
Redwood Shores, CA 94065
(415) 506-6160

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

Paper #3308

ASCII or EBCDIC: Only Your Printer Knows for Sure

Erik J. Vistica

ROLM
4900 Old Ironsides Drive
Santa Clara, CA 95054
(408) 492-5658

Introduction

Would you believe that sending reports from your HP cpu to a printer attached to an IBM mainframe could be as easy as:

```
:FILE HPREPORT;DEV=LPIBM
```

Easy, that is, once the process is set up.

The process consists of taking an ordinary spoolfile, determining its destination on the IBM, translating the carriage control from HP codes to the ANSI (American National Standards Institute) chars, packaging it in an IBM MVS job, and using NRJE (Network Remote Job Entry) to submit it to the IBM. The end result is that any IBM printer that you can get to from your IBM cpu can be accessed from your HP as though it were directly attached to the HP. What we are doing, in effect, is writing our own Spooler process.

Note: Please see the paper *Data Transfer with NRJE: Making Your Data Turn 'Blue'*, in these proceedings, for more details on NRJE. It is assumed that the reader is familiar with NRJE and what it does.

The process was written for Classic HPs and makes use of a third party network spooling product called Unispool. I do not suggest that Unispool is the best tool for the job. There are others that may work as well. It was simply the tool at hand so some of the examples are for that product. It would be possible to get the job done without it and still retain most of the benefits. I hear that the Native Mode Spooler would probably make things somewhat simpler as well.

The process also uses MPEX command files. MPEX may or may not be needed on MPE/iX.

HP Disc File to IBM Printer

Before we begin, let's see if we can send a simple HP disc file to an IBM printer. This is essentially the final step in our 'Spooler' process. It is important that you prove to yourself that this works, as it is a basic test that you will need when troubleshooting. We won't worry about how the output looks on the IBM printer at this point (page breaks will most likely look close, but not quite right, and a few characters will show up as some other symbol). We will discuss making it look right later. Just see if you get something resembling your original file for now.

The IBM printer that we will use first is an IBM 3800 laser printer configured on the IBM as class 'A'. In our case, this is analogous to an HP2680 laser printer on ldev 6. Since we are using NRJE, reports are sent to the IBM as instream data within an MVS job. On the HP, we keep the IBM JCL separate from our report. We use NRJE to send the two files with:

```
:NRJE RMT1
RMT1> Submit IBMJCL, HPREPORT; Maxrec=248
RMT1> Exit
:
```

Maxrec of 248 is used since this is the maximum record length that will fit through the link (or 252 depending on your config). We use the maximum as we may have reports wider than 132 characters.

The file IBMJCL:

```
//IBMPRINT JOB (,B24),'Test HP-IBM print',CLASS=A,MSGCLASS=Z
//*
//JS010     EXEC  PGM=IEBGENER
//SYSIN     DD DUMMY
//SYSPRINT  DD SYSOUT=*
//*
//OUT1      OUTPUT CLASS=A,DEST=LOCAL
//SYSUT2    DD SYSOUT=(,),OUTPUT=(*.OUT1),
//             DCB=(BLKSIZE=248,RECFM=UA)
//SYSUT1    DD DATA
```

The file HPREPORT is your report as a disc file.

The above example will work as is. The actual production JCL is discussed later.

Now that we know we can print an HP disc file on an IBM printer, let's look at how spoolfiles are moved within an HP network.

Network Spooling Basics

Say that you have a single HP cpu. If two applications, such as AP and Payroll, use the same printer, I strongly suggest that you define a separate class name for each one (LPAP and LPPAYROL). This lets each application reference the printer with its own name. Then, if the Payroll dept moves to another building and needs their own printer, you can easily move the LPPAYROL class to the new ldev and leave LPAP where it is. If you used only one class name (like DEPTLP), then your Payroll application programmer may not appreciate having to change the programs to reference a new class name (unless it is a well written application with the printer class name stored in one central place).

Now consider the multiple cpu environment. At our site, we had several HPs, but wanted to use a single NRJE link. For a period of time, we dedicated a Series 70 as a network hub. All HP printers were attached to this cpu. The NRJE line was here, and no users logged on. Unispool was used, on all 'leaf' nodes, to send all non-deferred spoolfiles to the proper printer on this hub node. Since no printers existed on any of the other HPs, application programmers were forced to use class names, rather than hardcoded ldevs, to reach the printers. Class names are required by the Unispool config so that it can map the source class name on the local cpu to the target class name on the remote cpu. (It is a good idea to NEVER use hardcoded ldevs in your applications anyway, although some systems software may require them due to MPE command syntax). Using class names allows you to configure, and later move, your printers to any ldev without changing your applications. Simply move the class name to the new ldev. (But you already do this, right?)

Before we could implement our hub, we had to convert one application that not only used ldevs, but had them stored in a database in an integer field (worse than simple hardcoding). What we did, was to leave the database as is (and the ldev numbers in it). For each ldev entry in the database, we created a class name consisting of the letter 'A' followed by the ldev (ldev 32 becomes class A32). Then we altered all programs that used ldevs from the database to place the letter 'A' before the ldev in the FOPEN (or whatever) call. The programs were now opening files by class instead of ldev. (This method was chosen since altering the database and programs to use character fields would have required much more work). I repeat 'NEVER USE LDEVS IN APPLICATIONS'! The rest of our applications already used class names.

In order to process spoolfiles with Unispool, you need to have a spool queue for your programs to write spoolfiles to, yet you don't want these spoolfiles to print. The makers of Unispool solve this by having you configure (in MPE) an ATP port as a serial printer (driver HIOASLP0) but not connect anything to it. You then issue a `:STOPSPOOL ldev;OPENQ`, or simply `:OPENQ ldev` if the device is not initially spooled, to stop the spooler process but leave the spool queue open. This creates the situation we want but is wasteful of expensive hardware. I suggest that you configure an HP2566 line printer (driver HIOCIPR0) on the same IMB and GIC as your tape drive but on an unused HPIB address (like DRT 23). (See the Appendix for sample MPE config). This creates a spool queue without wasting a physical port. I asked the makers of Unispool if there was any problem doing it this way. They said that it works fine but they felt that it makes the installation of their product more complex for their customers (I beg to differ).

At our site, we have two of these fake HP2566s configured. One on ldev 998, the other on ldev 999.

Our class naming convention consists of xPxxxxxx, xTxxxxxx, and xSxxxxxx classes. The xPxxxxxx classes are all on ldev 999. These are the class names that applications are allowed to use. (All of them are type 4 devices in Unispool). We also have Annn classes for the Hardcoded ldev workaround. We chose 'A' as the first letter so that these exception classes would sort first on a Sysinfo list.

Annn	Work around for hardcoded ldevs.
A32	Class for hardcoded ldev 32 (routed to class LPAP in Unispool).
LPxxxxxx	HP Line Printer (could be character printer or laserjet).
LP	HP system Line Printer.
LPAP	HP Line Printer for AP application.
LPPAYROL	HP Line Printer for Payroll application. If AP and Payroll use the same printer, this is routed to class LPAP within Unispool.
PP	HP system Page Printer (a 2680)
LASER	Alias for class PP (routed to class PP within Unispool).
IPA	IBM system Printer on class A.
IPMGRSYS	IBM Printer near the System Manager.

The xTxxxxxx (Transport) classes are all on ldev 998 (with an ACD to prevent applications from accessing them directly). There is one for each corresponding xSxxxxxx class. They are a workaround for limitations in Unispool (they are type 5 devices for transport to hub cpu, or type 4 if this is the hub).

LT	From class LP.
LTAP	From class LPAP.
PT	From class PP.
ITA	From class IPA and IPMGRSYS. All IPxxxxxx classes are routed to this class.

The xSxxxxxx (Spooler) classes exist only on the hub node and each is on its own physical printer ldev. Applications should NEVER reference these directly. Doing so would undermine the entire reason for moving the xPxxxxxx classes off of the physical printer ldev in the first place.

LS	On ldev 6. From class LT.
LSAP	On ldev 32 (for example). From class LTAP.
PS	On ldev 19. From class PT.
ISA	On ldev 998. From class ITA. Used by all IBM bound spoolfiles.

All of the LPxxxxxx classes are either routed to another LPxxxxxx class (in the case of alias names or shared printers) or to their corresponding LTxxxxxx class on the local node, then to their corresponding LSxxxxxx class on the hub node.

All of the IPxxxxxx classes are either routed to another IPxxxxxx class (for alias or shared) or to the single class name ITA on the local node, then to the class ITA on the hub node (I'll explain why later), and then on to class ISA (also on the hub node, ldev 998). All reports destined for the IBM end up with class ISA before they leave the HP network. This allows us to create our own 'Spooler' process, on the hub, that watches for spoolfiles on class ISA. Using this method, however, we can no longer determine the destination printer from the class name.

Retaining Destination Information

In order to retain the original destination, yet still have a common class name (ISA), we chose to embed the destination into the spoolfile itself. The alternative is to configure an ITxxxxxx and ISxxxxxx class for each IBM printer (no thanks, we already configured IPxxxxxx classes for each). The class ISA can be thought of as the queue for all IBM bound print files. The information is embedded into the spoolfile via a Unispool control procedure attached to each of the IPxxxxxx devices:

```
PROCEDURE WRITEVAR
WRITE ".MPEXVARMARKER"
WRITE "NODE = '!SYSTEM!'"
WRITE ".MPEXVARMARKER"
WRITE "BIN = '!PARM1!'"
WRITE ".MPEXVARMARKER"
WRITE "DEV = '!DEV!'"
WRITE ".MPEXVARMARKER"
WRITE "FORM = '!FORM!'"
END
```

The destination is known by the DEV=!DEV! (device class). The '.MPEXVARMARKER' records are a flag to a program in a monitor job on the hub node that processes spoolfiles. When the program sees this, it discards it, reads the next record, inserts 'SETVAR ' in front of it, and writes the record to a special file (more on this later). So, the information is passed to the hub node, but is stripped back out of the spoolfile.

'Virtual' Device Classes

Coldloading the system to add an IPxxxxxx class name every time I want to send reports to an as yet unconfigured IBM printer seems like work to me (not to mention the Unispool config additions as well). If we could find some way, other than class name, to specify the destination IBM printer, then we would not need to Coldload the system. If your programs create file equates within them, by using the Command intrinsic, and you do not want to alter them, then you must use a class name. View screens that prompt for class names for report destinations, and programs that retrieve class names from databases typically do this. If the class is specified in a file equate from a job or UDC, then we can use a common class name for all IBM bound output, yet still choose any IBM printer.

We do this using the Forms Message parameter of the :FILE command. The Forms Message parm is a free format, 49 character string, including the period. This gives us 48 characters to play with. To use it, we configured the class PARMFORM on ldev 999. Within the Unispool control procedure for class PARMFORM, the value of the Forms Message is written into the spoolfile (to be extracted later by our own Spooler process). Within the Forms Message, we specify the destination by:

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPPAYROL.
```

The '(' is our own syntax for Parameter Forms Messages. The 'D=' specifies that the destination is the 'Virtual' class name of IPPAYROL. (Note that there is NO class name IPPAYROL configured on any ldev.) A 'Virtual' class name is actually a file name. The file is an MPEX command file with SETVARs that specify the IBM CLASS, DESTination, and printer model. The printer model is used to decide defaults for fonts, etc. The CLASS and DEST are substituted into our IBM template job. (We'll talk more about the template job later). The Forms Message can be used to pass other parms as well.

MPEX command file for Virtual class IPA:

```
:COMMENT  FILE=IPA
:SETVAR  CLASS = 'A'
:SETVAR  DEST  = 'LOCAL'
:SETVAR  MODEL = '3800'
```

IBM Printer Features

At our site, we use 4 different models of IBM printers:

- 3800 The system laser printer.
- 3820 A laser printer near the users (with many fancy features).
- 3812 A laser printer near the users (with fewer features).
- 4245 A pin-feed printer (for special forms and green-bar paper).

The 38xx laser printers mentioned are capable of different fonts and Page Definitions (like HP Environment files). Pagedefs allow 1, 2 or 4 logical pages per physical page in various orientations. The 3820 can print on both sides of the paper (called Duplex). When using Duplex, the back side is printed so that you can bind your output on the left side in portrait mode or the top in landscape mode and it will look right. There is another form of printing on both sides called Tumble. This prints the back side so that you can bind the top in portrait or the left side in landscape.

Using our Parameter Forms Message, we defined one and two letter keywords so we could access these printer features.

Keyword	
C	The IBM CHARS keyword (the font). Used to override the default. Supply actual IBM font names here. This parm is seldom used since the default font is chosen depending on Orientation and logical pages per physical page.
D	Device. A 'Virtual' class name (actually a command file).
F	Forms. The IBM FORMS keyword (Form-ID). Used for special cases like special forms or to send output to RMDS (Report Management and Distribution System) on the IBM.
FD	FormDef. The IBM FORMDEF keyword. Used to specify Duplex or Tumble. Valid values are D, and T (Actual allowable IBM values are incomprehensible strings of numbers).
M	Msg. Our flag to tell the process to send you back a TELL msg once the job executes on the IBM.
N	Node. Our name for the IBM node when sending a file to a VMid.
O	Orientation. P for Portrait, L for Landscape. The default changes depending on the number of logical pages per physical page (L for 1 or 2 up; P for 4 up).
P	Pagedef. The IBM PAGEDEF keyword. Used to override our defaults. Supply actual IBM Pagedefs here (or P=2 for 2 up or P=4 for 4 up).
U	User. Our name for the IBM user when sending a file to a VMid.

To use these parms, you must use the PARMFORM class. If you just want the IBM defaults, and you have a specific class name configured for your IBM printer, you can use your specific class name directly.

To IBM 3800 class A using specific class name.

```
:FILE HPREPORT;DEV=IPA
```

To IBM 3800 class A using HP class PARMFORM (this produces the same results as the previous example).

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPA.
```

To IBM 3800 class A using Portrait Orientation.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPA,O=P.
```

To IBM 3820 near the System Manager, Landscape.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS.
```

To IBM 3820 with 2 logical pages, side by side, on each physical page. The physical page is in Landscape.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS,P=2.
```

To IBM 3820 with 2 logical pages, one over the other, on each physical page. The physical page is in Portrait.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS,P=2,O=P.
```

To IBM 3820 with 4 logical pages on one physical page. The physical page is in Portrait.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS,P=4.
```

To IBM 3820, print on both sides, bind on top, Landscape.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS,FD=D.
```

To IBM 3820, print on both sides, bind on left, Landscape.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS,FD=T.
```

To IBM 3820, print on both sides, bind on left, Portrait.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS,FD=D,O=P.
```

To IBM 3820, print on both sides, bind on left, Portrait, 4 logical pages per side (8 per sheet).

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPMGRSYS,FD=D,P=4.
```

To IBM 3812 in control room, Landscape.

```
:FILE HPREPORT;DEV=IPCTRLRM
```

To IBM 3812 in Portrait.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPCTRLRM,O=P.
```

To IBM VMid.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(N=XXXVM2,U=VISTICA.
```

To IBM RMDS.

```
:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPRMDS,F=X001.
```

HP Spoolfile to HP Discfile

Since most reports end up as spoolfiles, not disc files, we need a way to process spoolfiles with NRJE. Problem is, you can't. NRJE only works with disc files. If we could turn our spoolfile into a disc file, then NRJE could deal with it. What tool do we have to do this? How about SPOOK5. Using its Copy command and a File equate, you can create the kind of disc file you need.

The original implementation of this process did use SPOOK5 to create a disc file from a spoolfile. This disc file was then processed by our carriage control translation program. This worked for many of our reports but not all. SPOOK5 does not copy all of the carriage control information (like whether we are in pre-space or post-space mode). Taking an educated guess, based on the placement of page breaks, etc..., did not work for any and every spoolfile. We kept finding new ways that carriage control was done on the HP. Output from some BASIC programs are the worst. Quiz does things a certain way. LISTF doesn't do a page break before the first line. FORTRAN 66 doesn't do page breaks every 60 lines (it relies on the Automatic Perforation Skip feature of HP printers). PASCAL does things another way. COBOL can have both pre-space and post-space in the same file! And on, and on, ...

It is important to have ALL of the carriage control info because HP printers default to Post-space mode with Auto-Perf-Skip. IBM laser printers use Pre-space without Auto-Perf-Skip. Also, IBM printers do not understand all of those HP printer codes (see the HP Intrinsics manual under FWRITE). The IBM printers do understand the 5 ANSI chars. These are:

Table 1. ANSI CCTL Codes

Code	Action
blank	Single space
0	Double space
-	Triple space
+	No space (Carriage Return. No Linefeed)
1	Page eject (if not at Top-Of-Form)

The decision was made to bite the bullet and go after the raw carriage control data in the spoolfile itself (via a PM program). This PM program is on the swap tape and was written for Classics running Platform 1P. For an excellent look into the structure of spoolfiles, please see the article *Analysis of a Spoolfile* in the November 1988 issue of Interact.

ASCII to EBCDIC Translation

We mentioned earlier that not all characters in your HP disc file would translate to the same symbol on an IBM printer. Characters on the HP are encoded in ASCII while on the IBM they are in EBCDIC. The standard translation table appears in Appendix C of the FCOPY manual. If we just send our file as is, NRJE will do the translation for us using the standard table. You can do this, but, if your output has one of a small handful of certain characters in it, you are going to be disappointed with the results. This is because the standard table does not translate all characters to the same symbol in the other code. I have no idea as to why. There was probably a good reason at the time but that doesn't help us now. If we want all characters translated properly, we will have to define our own table and have a program call the CTRANSLATE intrinsic to translate the file ourself. The handful of characters I know of so far are:

`	Accent
!	Exclamation point
|	Vertical line
[	Left bracket
]	Right bracket
{	Left brace
}	Right brace

To make life a little easier writing the translation program, we defined our translation table to be standard ASCII, except for the 7 chars mentioned. Our program translates ASCII to ASCII, but, for the seven exceptions, it translates to the character that when translated again with the standard table, will become the desired character in EBCDIC (it confuses me too). Kind of like having the 7 chars take one step backward. We let NRJE do the actual translation to EBCDIC with the standard table. Please note that some of these characters may not look right on an IBM tube even though they look OK on an IBM printer (another mystery).

Template IBM Job

Each report transferred is sent as instream data in an IBM job. There is only one job used for all reports. This job is actually a template with variable fields. These variables are replaced by the appropriate values for the current report being sent. The template job is:

```
//&VIBMJOB JOB (,B&VBIN),'&VNAME',CLASS=J,MSGCLASS=S
//*
//*FILE = IBMJOB.NRJEPRT.SYS
//*
//*
&VTELLSTEP//* SEND TELL MSG BACK TO HP USER.
&VTELLSTEP//STEP1    EXEC  PGM=IEBGENER
&VTELLSTEP//SYSIN    DD DUMMY
&VTELLSTEP//SYSPRINT DD SYSOUT=*
&VTELLSTEP//*
&VTELLSTEP//SYSUT2   DD SYSOUT=(B,,MSGQ)
&VTELLSTEP//SYSUT1   DD DATA
&VTELLSTEP:FILE &VNODE.S&VJSNO.&VIBMJOB=TELLDUMY;DEV=LP&VNODENO;FORMS=S&VJSNO &V
&VTELLSTEP:FILE WORKARND=*&VNODE.S&VJSNO.&VIBMJOB
&VTELLSTEP:LISTF TELLDUMY.NRJEPRT.SYS;*WORKARND
&VTELLSTEP/*
//*
//*
//* PRINT FILE
//STEP2    EXEC  PGM=IEBGENER
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
//*
//OUT1     OUTPUT CLASS=&VCLASS,DEST=&VDEST,
//            COPIES=&VCOPIES,FORMS=&VIBMFORMS,
//            FCB=&VFCB,FORMDEF=&VFORMDEF,
//            PAGEDEF=&VPAGEDEF,CHARS=&VCHARS,
//            PRTY=&VPRTY
//SYSUT2   DD SYSOUT=(,),OUTPUT=(*.OUT1),
//            DCB=(BLKSIZE=248,RECFM=UA)
//SYSUT1   DD DATA
```

All strings with '&V...' are our variables that are replaced with the default values unless overridden by a Forms Message Parm. The job card specifies Class = J. This is a seldom used class at our site which means reports do not wait in the same job class (queue) as regular IBM jobs. The Msgclass = S does the same thing as :SET STDLIST=DELETE on the HP.

The VTELLSTEP section sends some HP commands back to the hub HP. One of these commands is a `:LISTF TELLDUMY.NRJEPRT;...` The file TELLDUMY.NRJEPRT is an empty file. We just need a filename that we know exists so we can do the :LISTF. We will discuss this feature in more depth later. Once all of the parms have been substituted, the job might look like:

```
//VISTICA JOB (,B24),'SYSINFO',CLASS=J,MSGCLASS=S
//*
//*FILE = IBMJOB.NRJEPRT.SYS
//*
//*
//* //* SEND TELL MSG BACK TO HP USER.
//* //STEP1    EXEC  PGM=IEBGENER
//* //SYSIN    DD DUMMY
//* //SYSPRINT DD SYSOUT=*
//* //*
//* //SYSUT2   DD SYSOUT=(B,,MSGQ)
//* //SYSUT1   DD DATA
:FILE CPU3.S39.VISTICA=TELLDUMY;DEV=LP03;FORMS=S39 VISTICA IN MVS OUTPUT QUEUE.
:FILE WORKARND=*CPU3.S39.VISTICA
:LISTF TELLDUMY.NRJEPRT.SYS;*WORKARND
/*
//*
//*
//* PRINT FILE
//STEP2    EXEC  PGM=IEBGENER
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
//*
//OUT1     OUTPUT CLASS=N,DEST=U15,
//            COPIES=1,FORMS=,
//            FCB=,FORMDEF=010111,
//            PAGEDEF=W12881,CHARS=GT15,
//            PRTY=
//SYSUT2   DD SYSOUT=(,),OUTPUT=(*.OUT1),
//            DCB=(BLKSIZE=248,RECFM=UA)
//SYSUT1   DD DATA
```

A Spoolfile's Journey

The journey of an HP spoolfile to an IBM printer begins with its creation. We configured (in MPE) the class name IPA (IBM Printer in print class A).

```
:FILE HPREPORT;DEV=IPA
:LISTF @,2;*HPREPORT
```

We configured Unispool to take spoolfiles from class IPA (type 4), write some info (like MPE class name) into the spoolfile, and move them to class ITA (IBM bound Transfer device for print class A).

We also configured Unispool (on the leaf nodes) to take spoolfiles from class ITA and transfer them to the hub node class ITA.

On the hub node, Unispool takes spoolfiles from class ITA, writes a dummy record to a message file (JPRTDFID.NRJEPRT.SYS), and leaves the spoolfile on class ISA 'as is' (We configged Unispool to ignore spoolfiles on class ISA).

Also on the hub, our Spool process monitor job (JPRTMON) is running using MPEX command files. The job begins by first putting itself to sleep by posting a Read to the message file JPRTDFID. When Unispool writes a dummy record to this file, the job wakes up, empties any other dummy records from the file, and starts its processing cycle. Once the cycle is done, the job starts this loop again by posting a Read to JPRTDFID, thus putting itself back to sleep. If Unispool has written more dummy records while we were in the processing cycle, the Read is satisfied immediately and the cycle begins again.

The processing cycle consists of scanning for all spoolfiles that are on class ISA, in a READY state, and with an OUTPRI greater than our MPEX VAR called PRTFENCE (needed because MPEX does not provide access to the current Outfence value. This also allows us to control IBM bound spoolfiles separately from HP printer bound spoolfiles).

For each spoolfile that qualifies, we run our special PM program. This program copies a spoolfile to a disc file while translating the carriage control to Pre-space ANSI, changing our 7 problem chars to the needed value for later EBCDIC translation, and stripping out the MPEXVARMARKER info as it is written to a separate disc file. We then use the MPEXVARMARKER info to alter the default settings, modify our template job, and Submit the job plus our report disc file to NRJE.

The job runs on the IBM and produces an IBM output spoolfile. This spoolfile routes through the IBM network to its final destination (an MVS printer, a VM printer, a VM User-ID, or RMDS).

Just think. All this work from a simple class name in a file equate.

Feedback Tell Message

If we chose to use the 'M' (for Message) parm in our Forms Msg, even more work occurs. This parm causes a certain section of the template job to have its Comments (//*) removed, thus enabling that section. This section causes three records (2 file equates and a :LISTF) to be sent back to the HP hub node. These records are sent back to Form-ID MSGQ (defined in the file NRJETABL) which is equated to the message file MSGQ. Within the job JPRTMON, there is a son process that runs concurrently with its MPEX father process. This son process watches the MSGQ file. When it sees data, it filters it (to be sure that security is maintained), then executes it. This causes a dummy spoolfile to be created that has a Forms Msg attached. The dummy spoolfile's device determines the originating HP's node. The Forms Msg carries the originating HP's session number and filename. Unispool, on the hub, forwards this dummy spoolfile to the source node that created the original report spoolfile. When Unispool, on the original source node, processes the dummy spoolfile, it creates and executes a :TELL command from the information in the Forms Msg, and finally, deletes the dummy spoolfile.

I tell you, that Forms Msg parm sure earns its keep.

Sending Reports and $STDLISTs to RMDS

RMDS (IBM's Report Management and Distribution System) is a facility that lets IBM users view reports online. To save printing costs, we decided to have RMDS handle some of our HP output. Within RMDS, on the IBM, you must define a Form-ID to report name relationship. Then, within your IBM job, you reference this Form-ID when creating your report, and send the output to the proper IBM class.

Suppose that we have 2 HPs which we'll call HP1 and HP2. We run the same job on both HP1 and HP2 that produces the report HPREPORT that we want routed to RMDS. First, have your IBM RMDS person define 2 Form-IDs to RMDS (let's suppose that they chose R101 and R201). Then you specify the respective Form-ID in the file equate in the job on each HP.

For HP1 `:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPRMDS,F=R101.`

For HP2 `:FILE HPREPORT;DEV=PARMFORM;FORMS=(D=IPRMDS,F=R201.`

That's all there is to it.

Now suppose that we run a nightly backup job on each HP called JBACKUP and want to send the $STDLIST for each to RMDS. The above file equates work fine for regular report output, but, try as we might, we can't specify Forms Msgs in HP Job cards.

In order to attach a Forms Msg to a $STDLIST spoolfile, we use the Unispool control procedure named in the lookup file entry for this jobname.

Lookup entry for HP1.

File	$STDLIST
Dev	@
Job	JBACKUP
User	@
Acct	@
Parm1	CALL
Parm2	JBACKUP_STDLIST
Parm3	R111

Lookup entry for HP2 (same as HP1 except Parm3):

Parm3	R211

The control procedure is:

```
PROCEDURE  JBACKUP_STDLIST
WRITE '.MPEXVARMARKER'
WRITE 'FORM="(D=IPRMDS,F=!PARM3!"'
END
```

HP Disc File to IBM VMid

Suppose that you want to send one of your HP disc files to your IBM VMid. If you use a PC running a terminal emulator to access both the HP and IBM, one approach is to start your HP terminal emulator, log on to the HP, download the file to your PC, stop the HP terminal emulator, start the IBM terminal emulator, log on to VM, and upload the file. Even if your baud rate is 9600, this can take a while for large files, tying up your PC for the duration.

Since an MVS job can send a file to a VMid, and we are using an MVS job from the HP, we can send a file all the way from the HP to VM. We already showed a sample file equate to route output to a VMid, but, to make it really simple, we created this UDC:

```
IBMSEND  FROMFILE=" ", TOFILE=" ", USER, NODE="XXXVM2", &
  MSG=" "

SETJCW  UDCPARM1BLANK = 0
SETJCW  UDCPARM1BLANK!FROMFILE = 1
SETJCW  UDCPARM2BLANK = 0
SETJCW  UDCPARM2BLANK!TOFILE = 1
SETJCW  UDCPARM5BLANK = 0
SETJCW  UDCPARM5BLANK!MSG = 1

IF   UDCPARM1BLANK = 1  THEN
     FILE FROMFILE=$STDINX
ELSE
     FILE FROMFILE=!FROMFILE
ENDIF

IF   UDCPARM2BLANK = 1  THEN
     IF   UDCPARM1BLANK = 1  THEN
          IF   UDCPARM5BLANK = 1  THEN
               FILE TOFILE=IBMSEND;DEV=PARMFORM; &
                 FORMS=(U=!USER,N=!NODE.
          ELSE
               FILE TOFILE=IBMSEND;DEV=PARMFORM; &
                 FORMS=(U=!USER,N=!NODE,M.
          ENDIF
     ELSE
          IF   UDCPARM5BLANK = 1  THEN
               FILE TOFILE=!FROMFILE;DEV=PARMFORM;&
                 FORMS=(U=!USER,N=!NODE.
          ELSE
               FILE TOFILE=!FROMFILE;DEV=PARMFORM;&
                 FORMS=(U=!USER,N=!NODE,M.
          ENDIF
     ENDIF
ELSE
     IF   UDCPARM5BLANK = 1  THEN
          FILE TOFILE=!TOFILE;DEV=PARMFORM;&
            FORMS=(U=!USER,N=!NODE.
     ELSE
          FILE TOFILE=!TOFILE;DEV=PARMFORM;&
            FORMS=(U=!USER,N=!NODE,M.
     ENDIF
ENDIF

FCOPY FROM=*FROMFILE;TO=*TOFILE;CHAR;NORECNUM
**********************************************************************
```

Now I can send the file HPREPORT to my VMid of Vistica on the default IBM node of XXXVM2 with:

```
IBMSEND  HPREPORT,,VISTICA
```

If I want the Tofile on the IBM to have a different name, and I want to send the file to the VMid of Smith on the node YYYVM1, and I want MVS to send a message back to the HP once it processes the file:

```
IBMSEND  HPREPORT, IBMREPT, SMITH, YYYVM1, M
```

It doesn't get much easier than this.

VM File to HP Disc File

Since there is a way to send files from HP to VM, I wondered if there was a way to send them back. Our VM support dept gave me the following 'Exec' to Submit a VM file to MVS for execution as a Job. I vaguely understand how it works, but, like so many things in life, understanding is not a prerequisite for effective use (do you really know how your microwave oven works, or do you just press the < Nuke-it > button).

```
/* punch a file to mvs1 */
Arg infile
If infile='' Then Do;
  Say 'no arguments supplied, nothing done.'
  Exit 24
  End
Parse var infile fn ft fm .
'IDENTIFY ( LIFO'
If rc=0 Then
  Pull userid . nodeid . rscsid .
  Else
  Pull rscsid='UNKNOWN'
'VMPUSH PUN'
'CP SPOOL PUNCH ' rscsid
'CP TAG DEV PUNCH XXXMVS1 JOB'
'PUNCH' fn ft fm '( NOHEADER'
'VMPOP  PUN'
Exit
```

It is your responsibility to front-end your data with the MVS JCL. You will also need HP JCL if you don't have a dedicated Form-ID in the file NRJETABL.NRJE.SYS. You could alternatively use the HP !DATA command followed by your data instead of an HP job. To send 4 data records to the HP file IBMDATA, your VM file might be:

```
//XXXEJV   JOB (,B99),'VM to HP',CLASS=J,MSGCLASS=Z
//*
//JS010    EXEC  PGM=IEBGENER
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT2   DD SYSOUT=(B,,CMD),DEST=RMT1
//SYSUT1   DD DATA
!JOB VMTOHP,MGR.TEST;OUTCLASS=,1
!BUILD IBMDATA
!FCOPY FROM;TO=IBMDATA
Data Record 1
Data Record 2
Data Record 3
Data Record 4
:EOD
!EOJ
```

WSIDs vs Links

If you have only one NRJE link, then all of your network traffic to your IBM travels through a single 'pipe'. If this link is used for sending large data files as well as small reports to be printed, your small reports could wait quite a while behind a large transfer.

You can avoid this by configuring another WSID on both the HP and in JES. Both WSIDs use the same physical link. Then have your large file transfer applications use the first WSID (RMT1) and have your small report transfers use the second (RMT2). NRJE will interleave the traffic for the two WSIDs so that the small transfer won't have to wait.

Sending IBM output to HP printers

Using the same concepts presented here, I believe that a similar process could be written to route IBM print files to HP printers. We did not attempt to write such a process, although we have succeeded in using the NRJE Lookup table to route output to HP printers (how to do this is fully documented in the HP NRJE Manuals).

Conclusion

There is no denying that this is a complex process that requires detailed knowledge in several diverse areas. The benefit is that only a limited few need to understand it all. Your users and programmers can take full advantage of all of your IBM printers using standard HP command syntax never even having cracked the cover on an IBM manual.

APPENDIX

Disclaimer

These examples have been used successfully by ROLM. However, NO warranty is made to the accuracy of the examples or any related material, and NO responsibility is assumed for any modification directly or indirectly caused by the use of these examples. It is the user's responsibility to evaluate the usefulness of the material supplied. Also, NO guarantee is made to provide any corrections or extensions made in the future.

Sample NRJE Lookup Table

Add this entry to the file NRJETABL.NRJE.SYS. Needed by the Tell back message feature.

```
MSGQ     =MSGQ.NRJEPRT,OLD;SHR;ACC=APPEND
```

Sample MPE Config

```
LOG DRT U  C T SUB                REC   OUTPUT  MODE    DRIVER   DEVICE
DEV  #  N  H Y TYPE  TERMINAL     WIDTH  DEV             NAME    CLASSES
 #      I  A P      TYPE SPEED
        T  N E
1   25  0  0 3  8                  128    0             HIOMDSC2 SYSDISC
                                                                 DISCN
                                                                 DISC1
2   145 0  0 3  10                 128    0             HIOMDSC2 DISC
                                                                 DISCN
                                                                 DISC2
                                                                 SPOOL
6   152 0  0 32 9                  66     0         S   HIOCIPR0 LS
                                                                 PRINTER
                                                                 LPHOT
7   17  0  0 24 5                  128    0             HIOTAPE3 TAPE
                                                                 DDUMP
8   18  0  0 24 5                  128    0          R  HIOTAPE3 TAPEAUTO
9   19  0  0 3  8                  128    0          R  HIOMDSC2 CDROM
10  20  0  0 24 0                  128   LP      JA     HIOTAPE0 JOBTAPE

:

998 22  0  0 32 9                  66     0         S   HIOCIPR0 LP01
                                                                 LP02
                                                                 LP03
                                                                 LT
                                                                 LTAP
                                                                 ISA
                                                                 ITA
                                                                 PRINTER
999 23  0  0 32 9                  66     0         S   HIOCIPR0 A32
                                                                 IPA
                                                                 IPAP
                                                                 IPZ
                                                                 LASER
                                                                 LP
                                                                 LPAP
                                                                 LPPAYROL
                                                                 PARMFORM
                                                                 PP
                                                                 PRINTER
```

Unispool Examples

Unispool Configs

Unispool config file for CPU3 (leaf node):

```
 M   CPU3;15;2/C;2;MM/DD/YY
*Remote systems
 S  1;CPU1 ;6;;;;;;;;I;CS
 N  1;CPU1 ;HP3000
 S  2;CPU2 ;6;;;;;;;;I;CS
 N  2;CPU2 ;HP3000
*HP System printers
 D 1  ;LP      ;4;;;;;;     ;LT      ;1;CS
 X 1  ;;HEADER;TRAILER
 D 2  ;PP      ;4;;;;;;     ;PT      ;I;CS
 X 2  ;;HEADER;TRAILER
*HP System printer aliases
 D 3  ;LASER   ;4;;;;;;     ;PP      ;I;CS
 X 3  ;;NULL
*HP Remote printers
 D 10 ;LPAP    ;4;;;;;;     ;LTAP    ;I;CS
 X 10 ;;SAVE
*HP Remote printer aliases
 D 11 ;LPPAYROL;4;;;;;;     ;LPAP    ;I;CS
 X 11 ;;NULL
*IBM System printers (3800)
 D 101;IPA     ;4;;;;;;     ;ITA     ;I;CS
 X 101;;HEADER
 D 102;IPZ     ;4;;;;;;     ;ITA     ;I;CS
 X 102;;HEADER
*IBM Remote printers (3812/3820/Misc)
 D 110;IPAP    ;4;;;;;;     ;ITA     ;I;CS
 X 110;;IPREMOTE
 D 111;IPMGRSYS;4;;;;;;     ;ITA     ;I;CS
 X 111;;IPREMOTE
*IBM Special forms printer
 D 120;IPW     ;4;;;;;;     ;ITA     ;I;CS
 X 120;;IPREMOTE
*IBM Remote printer aliases
 D 131;LPMGRSYS;4;;;;;;     ;IPMGRSYS;I;CS
 X 131;;NULL
 D 132;IPPAYROL;4;;;;;;     ;IPAP    ;I;CS
 X 132;;NULL
*IBM - Use forms msg for passing parms to Unispool.
 D 140;PARMFORM;4;;;;;;     ;ITA     ;I;CS
 X 140;;PARMFORM
*For sending spoolfiles to any cpu.
 D 601;LP01    ;5;;;;;;CPU1 ;LP01
 D 602;LP02    ;5;;;;;;CPU2 ;LP02
 D 603;LP03    ;4;;;;;;     ;LS      ;I;CS
 X 603;;TELLDUMY
*Workaround for Unispool limitations
 D 706;LT      ;5;;;;;;CPU1 ;LS
 D 719;PT      ;5;;;;;;CPU1 ;PS
 D 732;LTAP    ;5;;;;;;CPU1 ;LSAP
 D 798;ITA     ;5;;;;;;CPU1 ;ITA
*End of config
```

Major differences for CPU1 (hub node):

```
*For sending spoolfiles to any cpu.
 D 601;LP01    ;4;;;;;;     ;LS      ;I;CS
 X 601;;TELLDUMY
 D 602;LP02    ;5;;;;;;CPU2 ;LP02
 D 603;LP03    ;5;;;;;;CPU3 ;LP03
*Workaround for Unispool limitations
 D 706;LT      ;4;;;;;;     ;LS      ;I;CS
 X 706;;NULL
 D 719;PT      ;4;;;;;;     ;PS      ;I;CS
 X 719;;NULL
 D 732;LTAP    ;4;;;;;;     ;LSAP    ;I;CS
 X 732;;NULL
 D 798;ITA     ;4;;;;;;     ;ISA     ;I;CS
 X 798;;ITA
*Physical printers
 D 906;LS      ;0;;6
 D 919;PS      ;0;;19
 D 932;LSAP    ;0;;32
*IBM bound spoolfile queue
 D 998;ISA     ;0;;998
*End of config
```

Unispool Control Procedures

```
PROCEDURE HEADER

COMMENT  PARM1 = nnn (THE BIN NUMBER)
COMMENT          CALL Parm2 as control proc
COMMENT          DISTRIB use Parm2 as distrib list
COMMENT          HEADOFF suppress unispool banners
COMMENT  PARM2 = banner text, or control proc, or distrib list
COMMENT  PARM3 = banner text, or Form-ID for RMDS
COMMENT  PARM4 = banner text
COMMENT  PARM5 = # COPIES

SAVESPOOLFILE
LOOKUP

IF   DEV = 'IP@'
     CALL WRITEVAR
ENDIF

IF   PARM1 = 'DISTRIB'
     DISTRIBUTE !PARM2!
     DELETESPOOLFILE
     EXIT
ENDIF

IF   PARM1 = 'CALL'
     CALL !PARM2!
     EXIT
ENDIF


CALL STCOPIES

IF   DEV = 'LP'
     CALL ADDHDHP
ELSE
IF   DEV = 'PP'
     CALL ADDHDHP
ELSE
     CALL ADDHDIBM
ENDIF
ENDIF


IF   DEV = 'IP@'
     DELETEFORM
ENDIF

END
```

```
PROCEDURE WRITEVAR

WRITE ".MPEXVARMARKER"
WRITE "NODE = '!SYSTEM!'"
WRITE ".MPEXVARMARKER"
WRITE "BIN = '!PARM1!'"
WRITE ".MPEXVARMARKER"
WRITE "DEV = '!DEV!'"
WRITE ".MPEXVARMARKER"
WRITE "FORM = '!FORM!'"

END
```

```
PROCEDURE STCOPIES
IF   PARM5 # ''
     IF   PARM5 # '0'
          ALTSPOOLFILE COPIES=!PARM5!
     ENDIF
ENDIF
END
```

```
PROCEDURE ADDHDHP
COMMENT  ADD 2 HEADER PAGES FOR HP OUTPUT

IF   FORMS
     EXIT
ENDIF
IF   PARM1 = 'HEADOFF'
     EXIT
ENDIF

CALL BANNERH
NEW
CALL BANNERH
NEW

END
```

```
PROCEDURE BANNERH
CALL BANNER1
WRITE "HEADER PAGE"
CALL BANNER2
END
```

```
PROCEDURE BANNER1
COMMENT  THIS IS THE FIRST PORTION OF A BANNER PAGE
SET BOLDCHAR

LEFT
REPEAT 3
WRITE "!JOBNUM!; !FILENUM! * !JOBNAME!, !USER!.!ACCOUNT! * !DATE!,!TIME!"
SPACE 2
BOLD1 "!SYSTEM! !FILENAME!"
SPACE 3

END
```

```
PROCEDURE BANNER2
COMMENT  THIS IS THE SECOND PORTION OF A BANNER PAGE
SET BOLDCHAR

SPACE 4
CENTER
BOLD2 "BIN !PARM1!"
SPACE 2
BOLD1 "!PARM2!"
SPACE 2
BOLD1 "!PARM3!"
SPACE 2
BOLD1 "!PARM4!"

END
```

```
PROCEDURE ADDHDIBM
COMMENT  ADD 1 HEADER PAGE FOR IBM OUTPUT

IF   PARM1 = 'HEADOFF'
     EXIT
ENDIF

CALL BANNERH
NEW

END
```

```
PROCEDURE TRAILER

COMMENT  THE TEST 'PRI < 2' IS USED TO PREVENT TRAILERS FROM BEING
COMMENT  PRINTED ON FILES THAT WERE DEFERRED IN THE HEADER CONTROL
COMMENT  PROCEDURE BECAUSE OF SOME ERROR.
IF   PRI < 2
     EXIT
ENDIF

IF   PARM1 = 'DISTRIB'
     EXIT
ENDIF

CALL ADDTRAIL

END
```

```
PROCEDURE ADDTRAIL
COMMENT  ADD 1 TRAILER PAGE FOR HP OUTPUT
IF   FORMS
     EXIT
ENDIF
IF   PARM1 = 'HEADOFF'
     EXIT
ENDIF

NEW
CALL BANNERT
END
```

```
PROCEDURE BANNERT
CALL BANNER1
WRITE "TRAILER PAGE"
CALL BANNER2
END
```

```
PROCEDURE IPREMOTE
LOOKUP
SAVESPOOLFILE
CALL WRITEVAR
DELETEFORM
END
```

```
PROCEDURE PARMFORM

SAVESPOOLFILE
LOOKUP

IF   FORM # '(@'
     ALTSPOOLFILE DEV=PARMFORM;DEFER
     TELL "!JOBNUM! FORMS MSG FOR DEV=PARMFORM MUST BE '(@'"
     TELLOP "       FORMS MSG FOR DEV=PARMFORM MUST BE '(@'"
     TELL "!JOBNUM! ALTERED SPOOFLE !FILE!.!USER!.!ACCOUNT!;DEFER."
     TELLOP "       ALTERED SPOOFLE !FILE!.!USER!.!ACCOUNT!;DEFER."
     EXIT
ENDIF

CALL WRITEVAR
CALL FORMIPXX
DELETEFORM

END
```

```
PROCEDURE FORMIPXX

IF   FORM = '(D=IPA'
     CALL ADDHDIBM
     EXIT
ENDIF
IF   FORM = '(D=IPA,@'
     CALL ADDHDIBM
     EXIT
ENDIF
IF   FORM = '(D=IPZ'
     CALL ADDHDIBM
     EXIT
ENDIF
IF   FORM = '(D=IPZ,@'
     CALL ADDHDIBM
     EXIT
ENDIF

END
```

```
PROCEDURE ITA
COMMENT  I WAKE UP THE JOB JPRTMON.
MPE FILE TOFILE=JPRTDFID.NRJEPRT;SHR;ACC=APPEND
MPE RUN FCOPY.PUB;INFO="FROM=JPRTWAKE.NRJEPRT;TO=*TOFILE"
END
```

Create this single record file with Editor. Keep as JPRTWAKE.NRJEPRT.SYS:

```
:COMMENT I AM THE FCOPY FROMFILE FOR UNISPOOL PROC ITA. WAKE JPRTMON.
```

```
PROCEDURE TELLDUMY

COMMENT  CONVERT TELLDUMY FILES TO TELL MSG.
IF   FILENAME = 'TELLDUMY'
     IF   ACCOUNT = 'SYS'
          TELL '!FORM!'
          DELETESPOOLFILE
          EXIT
     ENDIF
ENDIF
END
```

```
PROCEDURE SAVE
SAVESPOOLFILE
END
```

```
PROCEDURE NULL
COMMENT  I EXIST BECAUSE UNISPOOL INSISTS THAT TYPE 4 DEVICES HAVE
COMMENT  A HEADER OR TRAILER PROC.
END
```

Spooler process Job and Command files

```
!JOB JPRTMON,NRJEJOBQ.SYS,NRJE;OUTCLASS=IPA,1;PRI=CS;INPRI=12
!
!COMMENT  I WAKE UP WHEN UNISPOOL WRITES A DUMMY REC TO THE MSG FILE
!COMMENT  JPRTDFID.NRJEPRT.SYS.
!COMMENT  I PROCESS ALL SPOOFLES ON DEV=ISA AND PRI > PRTFENCE AND
!COMMENT  STATUS = READY.
!COMMENT  THE PROCESS CHGS CCTL FROM HP TO ANSI THEN PACKAGES THE
!COMMENT  FILE IN AN IBM JOB AND SENDS IT TO THE IBM FOR PRINTING.
!
!TELL LOGECHO.SYS JPRTMON - LOGON COMPLETED.  START MPEX...
!
!RUN MPEX.PUB.VESOFT;INFO="XEQ JPRTMAIN.NRJEPRT"
!
!TELL LOGECHO.SYS JPRTMON - LOGOFF.
!
!EOJ
```

```
:COMMENT  FILE=JPRTMAIN.NRJEPRT

:SETVAR  MPEXCMDTRACE = 0
:SETVAR  HPAUTOCONT = TRUE

 XEQ JPRTMON.AACONFIG
:!TEL !HPDATEF !HPTIMEF  - JPRTMON       MPEX ACTIVATED.

:COMMENT  PREVENT 2 CONCURRENT RUNS OF THIS JOB.
:IF   JSCOUNT ('JPRTMON,NRJEJOBQ.SYS') > 1  THEN
      RETURN
:ENDIF

:COMMENT  GET CONSOLE CAPABILITIES.
:RUN SETCON5.UTILPRIV.SYS

:COMMENT  START MSGQ MONITOR PROCESS.
:RUN MPEX.PUB.VESOFT;GOON;INFO="XEQ JPRTMSGQ.NRJEPRT"


:COMMENT  FOR PROGRAM CCTLANSI.NRJEPRT
:FILE INPUTF,OLDTEMP
:FILE OUTPUTF=$NEWPASS;REC=-248,100,F,ASCII;NOCCTL;DISC=100000,32


:COMMENT  SCAN FOR FILES TO PROCESS.
:SETVAR  INFINITE = 1
 WHILE   INFINITE = 1
   XEQ JPRTMON.AACONFIG

:  REPEAT
     XEQ JPRTFILE.NRJEPRT  ![SPOOL.SPOOLFILENUM]
   FORFILES &
     @.@.@ (SPOOL.DEVICE='ISA' AND SPOOL.ISREADY AND &
            SPOOL.OUTPRI > !PRTFENCE):SPOOL

:  !TEL !HPDATEF !HPTIMEF  - JPRTMON       MSG WAIT.
   USE JPRTDFID.NRJEPRT
:  COMMENT  BUILT WITH:  BUILD JPRTDFID;REC=-80,1,F,ASCII;MSG
 ENDWHILE

:COMMENT  END JPRTMAIN
```

```
:COMMENT  MPEX CMD FILENAME = JPRTMON.AACONFIG
:SETVAR  WSID = 'RMT102'
:SETVAR  PRTFENCE = 5
:SETVAR  TEL = 'TELL LOGECHO.SYS'
```

```
:COMMENT  MPEX CMD FILENAME = JPRTMSGQ
:COMMENT  I AM CALLED BY JPRTMAIN AS A SON PROCESS.  I WAIT FOR
:COMMENT  DATA IN MSGQ.NRJEPRT AND PROCESS IT.

:!TEL !HPDATEF !HPTIMEF  - JPRTMON      MSGQ.MPEX UP.
:COMMENT  BUILD MSGQ.NRJEPRT;MSG;REC=-255,1,F,ASCII

:COMMENT  START INFINITE LOOP.  THIS SON WILL QUIT WITH THE FATHER
:SETVAR  MSGQ_INFINITE = 1
 WHILE   MSGQ_INFINITE = 1
:  !TEL !HPDATEF !HPTIMEF  - JPRTMON      MSGQ.MSG WAIT.
:  RUN FCOPY.PUB;NEW;INFO="FROM=MSGQ.NRJEPRT;TO=MSGQCOPY.NRJEPRT
:  !TEL !HPDATEF !HPTIMEF  - JPRTMON      MSGQ.EDIT.
:  RUN EDITOR.PUB;NEW;KILL;STDIN=MSGQEDIT.NRJEPRT
 ENDWHILE

:COMMENT  END JPRTMSGQ
```

```
<< EDITOR STDIN FILENAME = MSGQEDIT.NRJEPRT.SYS >>

<< FILTER EACH RECORD LOOKING FOR VALID :FILE AND :LISTF CMDS. >>
<< IF ANY INVALID, KEEP AS MSGQBAD >>
<< IF ALL VALID, KEEP AS MSGQXEQ THEN 'USE' >>

SET TIMES=1023

TEXT MSGQCOPY.NRJEPRT,UNN
WHILE FLAG
  BEGIN
  FIND ":FILE MIS"/*(LAST)
  NOT;OR
    BEGIN
    Q"BAD :FILE MIS"
    KEEP MSGQBAD.NRJEPRT,UNN
    SET TIMES=1
    END
  OR
    BEGIN
    FIND "=TELLDUMY;DEV=LP"/*(LAST)
    NOT;OR
      BEGIN
      Q"BAD =TELLDUMY;DEV=LP"
      KEEP MSGQBAD.NRJEPRT,UNN
      SET TIMES=1
      END
    OR
      BEGIN
      FIND *+1
      NOT;OR
        BEGIN
        Q"NO :FILE WORKARND"
        KEEP MSGQBAD.NRJEPRT,UNN
```

```
          SET TIMES=1
          END
        OR
          BEGIN
          FIND ":FILE WORKARND=*MIS"/*(LAST)
          NOT;OR
            BEGIN
            Q"BAD :FILE WORKARND"
            KEEP MSGQBAD.NRJEPRT,UNN
            SET TIMES=1
            END
          OR
            BEGIN
            FIND *+1
            NOT;OR
              BEGIN
              Q"NO :LISTF"
              KEEP MSGQBAD.NRJEPRT,UNN
              SET TIMES=1
              END
            OR
              BEGIN
              FIND ":LISTF TELLDUMY.NRJEPRT.SYS;*WORKARND"/*(LAST)
              NOT;OR
                BEGIN
                Q"BAD :LISTF"
                KEEP MSGQBAD.NRJEPRT,UNN
                SET TIMES=1
                END
              OR
                BEGIN
                FIND *+1
                FIND *+1
                NOT;OR
                  BEGIN
                  Q"FOUND EOF"
                  KEEP MSGQXEQ.NRJEPRT,UNN
                  USE  MSGQXEQ.NRJEPRT
                  SET TIMES=1
                  END
                OR
                  YES
                END
              END
            END
          END
        END
      END
    END

EXIT
<< END MSGQEDIT >>
```

```
PARM  DFID

:COMMENT  MPEX CMD FILENAME = JPRTFILE
:COMMENT  I AM CALLED BY JPRTMAIN. I AM CALLED ONCE FOR EACH SPOOFLE

:COMMENT  PROCESS SPOOLFILE USING MODIFIED TRANSLATION
:COMMENT  TABLES SO THAT WHEN NRJE TRANSLATES FROM ASCII
:COMMENT  TO EBCDIC, THE DESIRED CHAR WILL PRINT ON IBM.
:COMMENT  ALSO CONVERT CCTL TO ANSI-STANDARD CHARS.
:COMMENT  AND INSERT FORMFEEDS IF NEEDED.
:!TEL !HPDATEF !HPTIMEF  - JPRTMON       RUN CCTLANSI.
:PURGE           INPUTF,TEMP
:RENAME $OLDPASS,INPUTF,TEMP
:PURGE MPEXCMD,TEMP
:BUILD MPEXCMD;TEMP;REC=-80,3,F,ASCII;DISC=10,1
:FILE  MPEXCMD,OLDTEMP;TEMP;SHR;NOLOCK
:RUN CCTLANSI.NRJEPRT;PARM=!DFID;INFO="IBMPRINT"

:COMMENT  SETVARS AND CHOOSE DRIVER FILE.
 XEQ SETVARS.NRJEPRT

:!TEL !HPDATEF !HPTIMEF  - JPRTMON       SUBMIT THRU NRJE.
:SETJCW  CIERROR = OK
:RUN NRJE.NRJE;STDLIST=$STDLIST;INFO='!WSID';&
   INPUT='SUBMIT IBMJOBV.NRJEPRT, *DRIVER, $OLDPASS;MAXREC=248';&
   INPUT='EXIT'

:IF   CIERROR = OK  THEN
:     DELETESPOOLFILE #O!DFID
:ENDIF

:COMMENT  END JPRTFILE
```

```
:COMMENT  MPEX CMD FILENAME = SETVARS

:COMMENT  SET DEFAULTS
:SETVAR  BIN = 'HP'
:SETVAR  CHARS = ''
:SETVAR  CLASS = 'A'
:SETVAR  COPIES = ![SPOOL.NUMCOPIES]
:SETVAR  DEV = 'IPA'
:SETVAR  DEST = 'LOCAL'
:SETVAR  DRIVER = 'NULLSRC'
:SETVAR  FCB = ''
:SETVAR  FORM = ''
:SETVAR  FORMDEF = ''
:SETVAR  IBMFORMS = ''
:SETVAR  IBMJOB = '![SPOOL.JSNAME]'
:IF   IBMJOB = ''  THEN
:     SETVAR  IBMJOB = '![SPOOL.USER]'
:ENDIF
:SETVAR  NAME = '![SPOOL.FILE]'
:SETVAR  NODE = 'MIS6'
:SETVAR  IBMNODE = ''
:SETVAR  IBMUSER = ''
:SETVAR  MODEL = '3800'
:SETVAR  ORIENT = ''
:SETVAR  PAGEDEF = ''
:SETVAR  PRTY = ''
:SETVAR  TELLMSG = FALSE
:SETVAR  TELLSTEP = '//* '

:SETVAR  NODENO = INTEGERPARSE (NODE [3:2])
:IF   NODENO <= 9  THEN
:     SETVAR  NODENO = '0!NODENO'
:ENDIF


:COMMENT  PARSE THE FORMS MSG VARS.
:IF   FORM <> ''  THEN
      XEQ PARSFORM.NRJEPRT

:     IF   IBMNODE <> '' &
      OR   IBMUSER <> ''  THEN
:          SETVAR  IBMJOB = NAME
:     ENDIF

:     IF   TELLMSG = TRUE &
      AND  ![SPOOL.JOBTYPE] <= 1  THEN
:          SETVAR  TELLSTEP = ''
:     ENDIF
:ENDIF


:COMMENT  SET VARS BASED ON DEVICE.
:IF   DEV <> 'IP'  THEN
```

```
      XEQ !DEV.NRJEPRT
:ENDIF

:COMMENT  SET VARS BASED ON DEVICE RULES.
 XEQ DEVRULES.NRJEPRT


:COMMENT  POINT TO DRIVER FILE.
:FILE DRIVER=!DRIVER.NRJEPRT,OLD


:COMMENT  MODIFY JOB SOURCE.
:!TEL !HPDATEF !HPTIMEF  - JPRTMON      MODIFY JOB SOURCE.
:RUN EDITOR.PUB;STDLIST=$STDLIST;INMSG;NOACTIVATE
:SETVAR  EDITPIN = MPEXPIN

:RUN !EDITPIN;GOON;INMSG;INPUT="T IBMJOB.NRJEPRT;&
CQ\&VBIN\,\!BIN\,ALL;&
CQ\&VCHARS\,\!CHARS\,ALL;&
CQ\&VCLASS\,\!CLASS\,ALL;&
CQ\&VCOPIES\,\!COPIES\,ALL;&
CQ\&VDEST\,\!DEST\,ALL"

:RUN !EDITPIN;GOON;INMSG;INPUT="&
CQ\&VFCB\,\!FCB\,ALL;&
CQ\&VFORMDEF\,\!FORMDEF\,ALL;&
CQ\&VIBMFORMS\,\!IBMFORMS\,ALL;&
CQ\&VIBMJOB\,\!IBMJOB\,ALL"

:RUN !EDITPIN;GOON;INMSG;INPUT="&
CQ\&VNAME\,\!NAME\,ALL;&
CQ\&VPAGEDEF\,\!PAGEDEF\,ALL;&
CQ\&VPRTY\,\!PRTY\,ALL;&
CQ\&VNODENO\,\!NODENO\,ALL;&
CQ\&VNODE\,\!NODE\,ALL"

:RUN !EDITPIN;INMSG;INPUT="&
CQ\&VTELLSTEP\,\!TELLSTEP\,ALL;&
CQ\&VJSNO\,\![SPOOL.JOBNUMBER]\,ALL;&
K IBMJOBV.NRJEPRT,UNN;E"

:COMMENT  END SETVARS
```

```
:COMMENT  MPEX CMD FILENAME = PARSFORM
:COMMENT  I PARSE THE FORMS MSG VARS.

:COMMENT  STRIP OFF LEADING CONSTANT.
:SETVAR  FORM = FORM - '('

:SETVAR     PARSFORM_DONE = FALSE
:WHILE  NOT PARSFORM_DONE
:     SETVAR  TOKEN = TOKEN (FORM, ',')
:     IF   TOKEN [0:2] = 'D='  THEN
:          SETVAR  DEV = TOKEN [2:8]
:     ELSE
:     IF   TOKEN [0:2] = 'C='  THEN
:          SETVAR  CHARS = TOKEN [2:8]
:     ELSE
:     IF   TOKEN [0:2] = 'F='  THEN
:          SETVAR  IBMFORMS = TOKEN [2:8]
:     ELSE
:     IF   TOKEN [0:3] = 'FD='  THEN
:          SETVAR  FORMDEF = TOKEN [3:6]
:     ELSE
:     IF   TOKEN [0:1] = 'M'  THEN
:          SETVAR  TELLMSG = TRUE
:     ELSE
:     IF   TOKEN [0:2] = 'N='  THEN
:          SETVAR  IBMNODE = TOKEN [2:8]
:     ELSE
:     IF   TOKEN [0:2] = 'O='  THEN
:          SETVAR  ORIENT = TOKEN [2:8]
:     ELSE
:     IF   TOKEN [0:2] = 'P='  THEN
:          SETVAR  PAGEDEF = TOKEN [2:6]
:     ELSE
:     IF   TOKEN [0:2] = 'U='  THEN
:          SETVAR  IBMUSER = TOKEN [2:8]
:     ENDIF
:     ENDIF
:     ENDIF
:     ENDIF
:     ENDIF
:     ENDIF
:     ENDIF
:     ENDIF
:     ENDIF

:     SETVAR  FORM = REMTOKEN (FORM, ',')
:     IF   FORM = ''  THEN
:          SETVAR  PARSFORM_DONE = TRUE
:     ENDIF
 ENDWHILE
:COMMENT  END PARSFORM
```

```
:COMMENT  MPEX CMD FILENAME = DEVRULES

:IF   IBMNODE = ''  THEN
:     COMMENT  REAL PRINTER

:     IF   MODEL = '3800'  THEN
           XEQ RULE3800.NRJEPRT
:     ELSE
:     IF   MODEL = '3820'  THEN
           XEQ RULE3820.NRJEPRT
:     ELSE
:     IF   MODEL = '3812'
           XEQ RULE3812.NRJEPRT
:     ENDIF
:     ENDIF
:     ENDIF


:     COMMENT  SET FORMDEF.
:     IF   FORMDEF = 'D'  THEN
:          SETVAR  FORMDEF = '010111'
:     ELSE
:     IF   FORMDEF = 'T'  THEN
:          SETVAR  FORMDEF = '010112'
:     ELSE
:          IF   MODEL = '3820'  THEN
:               SETVAR  FORMDEF = '010110'
:          ENDIF
:     ENDIF
:     ENDIF
:ELSE
:     COMMENT  NODE.USER
:     SETVAR  DEST = '!IBMNODE.!IBMUSER'
:     SETVAR  CLASS = 'A'
:ENDIF

:COMMENT  END DEVRULES
```

```
:COMMENT  MPEX CMD FILENAME = RULE3800

:IF   PAGEDEF = '2' &
 OR   PAGEDEF = '4'  THEN
:     IF   FORMDEF <> ''  THEN
:          SETVAR  DRIVER = 'DFORMFD3'
:     ENDIF

:     IF   ORIENT = 'P' &
      OR   ORIENT = 'PV'  THEN
:          SETVAR  PAGEDEF = 'M132C1'
:          SETVAR  CHARS   = 'GT20'
:     ELSE
:     IF   ORIENT = 'LV'  THEN
:          SETVAR  PAGEDEF = 'M120C0'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT20'
:          ENDIF
:     ELSE
:          SETVAR  PAGEDEF = 'M16080'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT12'
:          ENDIF
:     ENDIF
:     ENDIF
:ELSE
:IF   PAGEDEF = ''  THEN
:     IF   ORIENT = 'P'  THEN
:          SETVAR  PAGEDEF = '06061'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT12'
:          ENDIF
:     ENDIF
:ENDIF
:ENDIF

:COMMENT  END RULE3800
```

```
:COMMENT  MPEX CMD FILENAME = RULE3820

:IF   PAGEDEF = '2'  THEN
:     IF   ORIENT = 'P'  THEN
:          SETVAR  PAGEDEF = 'W120C0'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT20'
:          ENDIF
:     ELSE
:          SETVAR  PAGEDEF = 'W12881'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT15'
:          ENDIF
:     ENDIF
:ELSE
:IF   PAGEDEF = '4'  THEN
:     IF   ORIENT = 'L'  THEN
:          SETVAR  PAGEDEF = 'W240F1'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT24'
:          ENDIF
:     ELSE
:          SETVAR  PAGEDEF = 'W256C0'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT20'
:          ENDIF
:     ENDIF
:ELSE
:IF   PAGEDEF = ''  THEN
:     IF   ORIENT = 'P'  THEN
:          SETVAR  PAGEDEF = 'A06460'

:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT10'
:          ENDIF
:     ELSE
:          SETVAR  PAGEDEF = 'V066S1'
:          IF   CHARS = ''  THEN
:               SETVAR  CHARS   = 'GT15'
:          ENDIF
:     ENDIF
:ENDIF
:ENDIF
:ENDIF

:COMMENT  END RULE3820
```

```
:COMMENT  MPEX CMD FILENAME = RULE3812

:IF   ORIENT = 'P'  THEN
:     SETVAR  DRIVER = 'D3812P'
:ELSE
:     SETVAR  DRIVER = 'D3812L'
:ENDIF

:COMMENT  END RULE3812
```

Driver Files

The 'driver' files are simply extra control information that is inserted in front of the actual report. They are used for sending extra Fromfeeds when doing 4 up printing and for sending the commands to a 3812 to put it in Portrait or Landscape mode.

Since the NRJE Submit command cannot use $NULL as one of the files, we create our own.

```
:BUILD NULLSRC.NRJEPRT;REC=-2,1,F,ASCII;DISC=1,1
```

The file D3812L is used for Landscape mode on a 3812. In both records, the first character is a space. In the second record, the second character is 'Escape' (esc).

```
 VM3812 J
 escSH CHAR 224;LMAR 0.5;PLEN 8.5;LPI 9.0;ORIENT W;OFF
```

The file D3812P is used for Portrait mode on a 3812. In both records, the first character is a space. In the second record, the second character is 'Escape' (esc).

```
 VM3812 J
 escSH LMAR 0.5;OFF
```

The file DFORMFD3 is used to insert 3 Formfeeds at the beginning of the report when using 4 logical pages per physical page.

```
1
1
1
```

Sample Virtual class command files

To System laser printer:

```
:COMMENT  FILE=IPA
:SETVAR  CLASS = 'A'
:SETVAR  DEST  = 'LOCAL'
:SETVAR  MODEL = '3800'
```

To Hold queue:

```
:COMMENT  FILE=IPZ.
:SETVAR  CLASS = 'Z'
:SETVAR  DEST  = 'LOCAL'
:SETVAR  MODEL = '3800'
```

To MVS printer on local node:

```
:COMMENT FILE=IPAP
:SETVAR  CLASS = 'N'
:SETVAR  DEST  = 'U15'
:SETVAR  MODEL = '3820'
```

To printer on VM:

```
:COMMENT FILE=IPPAYROL
:SETVAR  CLASS = 'A'
:SETVAR  DEST  = 'xxxVM2.P381204C'
:SETVAR  MODEL = '3812'
```

To special forms VM printer:

```
:COMMENT FILE=IPSTATMT
:SETVAR  CLASS = 'Q'
:SETVAR  DEST  = 'yyyVM1'
:SETVAR  MODEL = '4245'
```

To RMDS:

```
:COMMENT  FILE=IPRMDS
:SETVAR  CLASS    = 'R'
:SETVAR  DEST     = 'LOCAL'
:SETVAR  MODEL    = '3800'
```

Paper #3309

Data Transfer with NRJE: Making Your Data Turn 'Blue'

Erik J. Vistica

ROLM
4900 Old Ironsides Drive
Santa Clara, CA 95054
(408) 492-5658

Introduction

So you want to transfer data between HP and IBM cpus. Well, HP's Network Remote Job Entry (NRJE) may be just the means for you. NRJE is software on the HP that emulates an IBM 8100 RJE station. What is an 8100? I've never seen one but it has been described as a work station with a card reader (for sending data to the IBM), a card punch (for receiving data), and a printer (also for receiving data). I hear that the 8100 also has a keyboard, video screen, and disc (for spooling of card reading/punching and printer output).

As far as the IBM is concerned, the HP is not another computer. The IBM sees the HP as one of its peripherals. This means that communication is done from the IBM point of view. The IBM is either reading cards, punching cards, or sending print output.

Note: There is also a Console feature for sending console commands, and viewing the results, for an RJE station. This will not be discussed. Further information can be found in the HP NRJE manuals and the appropriate IBM operator manual for the Job Entry Subsystem (JES) that is used at your site.

This paper will not cover all aspects of NRJE (such as configuration). Rather, it will focus on the day to day use of NRJE (getting a file from here to there or there to here).

Most of the author's experience is on Classic HPs with just enough IBM to get by. The IBM JCL examples are for JES2.

All examples are for Series 70s running Platform 1P. MPEX command files are used. MPE command files should be sufficient on MPE/iX.

Why NRJE?

"What's wrong with tapes?", you may ask. Nothing. Then again, many things. Depends on your situation.

Tapes are cheap, simple to use, you probably already own them, and the drives, so no money to spend. I have found that tapes are a BIG 'warm fuzzy' for people who just don't trust those invisible bits in a wire. Tapes also provide a physical reminder to operators. If a tape has arrived, it is time to run the job. If an expected tape for input has not arrived by a certain time, you can easily tell (and go find the reason).

NRJE can be simple to use, but not always. You must purchase the software license for it, and SNA (Systems Network Architecture). You must buy MODEMs, cables, an INP, and configure both the HP and the IBM. The big payoff is that once it is setup, file transfers require no operator intervention.

As for which method is faster, only you can say. The fastest NRJE link is 56 KB. Tape I/O is much faster. But... Tapes have rewind time, and time for the operator to realize that the tape has unloaded. Time to dismount from one drive and mount on the other (This can be measured in hours if the tape needs to be sent overnight to a distant site). Then there is load time on the reading drive. Time for the operator to reply (if not using labelled tapes). And the possible error of mounting the wrong tape.

As you can see, either method, or even a mix of the two, can be considered 'best'.

A 'Simple' Transfer

When you send a file to the IBM, the IBM is simply reading a card deck. Not just any card deck, but a card deck that is a valid IBM job. So how do we structure our file (card deck) to contain our data as well as being an IBM job? Our file will be an IBM job that contains instream data that will be copied to a permanent file on the IBM. Before we look at an IBM job, let's see what the same concept would look like with an HP job.

```
!JOB INSTREAM,User/ABCD1234.Account;OUTCLASS=LP,1
!
!BUILD TOFILE
!FCOPY FROM;TO=TOFILE
Data record 1
Data record 2
Data record 3
Data record 4
:EOD
!
!EOJ
```

Simple enough. Just :STREAM this file and the 4 data records will be written to a file named TOFILE. To help in understanding the IBM JCL, the above job can be rewritten (line numbers are for later reference):

```
1  !JOB INSTREAM,User/ABCD1234.Account;OUTCLASS=LP,1
2  !
3  !FILE SYSPRINT=$STDLIST
4  !FILE SYSIN=$NULL          << Not used >>
5  !FILE SYSUT2=TOFILE,NEW;REC=-80,160,F,ASCII
6  !FILE SYSUT1=$STDINX
7  !RUN FCOPY.PUB.SYS;INFO="FROM=*SYSUT1;TO=*SYSUT2";STDLIST=*SYSPRINT
   Data record 1
   Data record 2
   Data record 3
   Data record 4
8  :EOD
9  !
10 !EOJ
```

An IBM job to do the same thing might look like the following (the line numbers are not part of the IBM job. They match up with the line numbers in the previous HP job example):

```
1  //INSTREAM JOB (,B99),'Erik Vistica',CLASS=A,MSGCLASS=Z,
1a //          USER=XXEJV,PASSWORD=ABCD1234
2  //*
7  //JS010    EXEC  PGM=IEBGENER
3  //SYSPRINT DD SYSOUT=*
4  //SYSIN    DD DUMMY
5  //SYSUT2   DD SYSOUT,DSN=XXEJV.TOFILE,
5a //            DCB=(LRECL=80,BLKSIZE=12800)
6  //SYSUT1   DD DATA
   Data record 1
   Data record 2
   Data record 3
   Data record 4
8  /*
9  //*
10 //
```

I will attempt to describe this IBM job in HP terms. Some IBM parms have no equivalent in HP.

Line 1 and 1a: The 'JOB' card. The '//' is the HP '!'. INSTREAM is similar to the Job/Session ID. B99 is the Bin number to be printed on the $STDLIST header/trailer (no HP equivalent). The programmer's name is next. CLASS=A is the streams device (A) that the job is submitted to (HPs only have one so no equivalent). MSGCLASS=Z is the output queue (Z) for the $STDLIST (IBM SYSLIST). Msgclass Z at our site is defined as the HOLD queue. This is the same as ;OUTCLASS=LP,1 (the $STDLIST is held in the spooler). USER is the same as User.Account (there are no IBM accounts). PASSWORD is hardcoded (more on this later).

Line 2 and 9: Empty comment lines. Same as '!' or '!COMMENT'.

Line 7: An 'EXEC' card. IEBGENER is an IBM program with functions similar to FCOPY and SUPRTOOL. The 'JS010' is the JOBSTEP label (no equivalent).

Line 3: A Data Definition (a DD statement). Like a file equate. Note that DD statements come after the EXEC cards that use them.

Line 4: Used to reference a file of program commands. Similar to redirecting $STDIN for FCOPY. The equivalent on the HP is unneeded.

Line 5 and 5a: For the TOFILE. Datasets (files) on IBMs are specified in reverse order of the HPs (like Account.Group.File). Also, you specify Block Size rather than Blocking Factor.

Line 6: Specifies that instream data follows this record. There is also a DD * syntax that acts very similar. See your IBM MVS JCL manual for details.

Line 8: Same as :EOD. This is not needed as EOD is assumed when End-Of-Job is reached.

Line 10: The !EOJ card. This is not needed as EOJ is assumed when End-Of-File is reached.

Now that we have a viable IBM job, we need to get the IBM to read it in and start execution. To do this, we must access the HP NRJE subsystem and put the card deck (file) into the card reader (NRJE spool queue). Assume that we have configured a WorkStation ID (WSID) on both the HP and in JES called RMT1. Our JCL file is called INSTREAM.

```
:NRJE RMT1
RMT1> Submit INSTREAM
RMT1> Exit
:
```

That's all that's needed to submit a job to the IBM. Since your JCL has been copied to a spoolfile, you may now even log off. The file will be transferred for you. The link does not need to be up when you do the submit either. When the link does come up, all READY non-deferred spoolfiles on the reader will be transferred.

Note: These spoolfiles can be looked at with SPOOK but since they have been translated from ASCII to EBCDIC (and have some extra control records in them) all you will see is garbage. More on translation later.

The NRJE Submit command can handle up to 5 files, not just one (See the HP NRJE manual for full details on NRJE commands). For example:

```
RMT1> Submit file1, file2, file3, file4, file5
```

All of the named files will be concatenated into one spoolfile. This makes it convenient to separate the JCL source file from the data. For example:

```
RMT1> Submit IBMJCL, HPDATA
```

Reverse NRJE

Ok, so we can send data to the IBM. How to send data to the HP? This is known as Reverse NRJE. The method is very similar. You compose an IBM job that copies instream data to an output file. The instream data, however, is actually a valid HP job and the output file is actually the card punch of the RJE Station (HP cpu). A sample IBM job to send a Tellop would be:

```
//TELLOP   JOB (,B99),'Some descriptive name',CLASS=A,MSGCLASS=Z
//*
//* NOTE: No USER or PASSWORD is needed since no permanent datasets (files)
//*       are accessed.
//*
//JS010    EXEC  PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2   DD SYSOUT=(B,,CMD),DEST=N5R1
//SYSUT1   DD DATA
!JOB TELLOP,User/ABCD1234.Account;OUTCLASS=LP,1
!TELLOP NRJE FROM IBM TO HP IS WORKING.
!EOJ
```

The 'SYSUT2 DD SYSOUT=(B,,CMD);DEST=N5R1' is what directs the output to the HP. The 'DEST=N5R1' directs the output to IBM node 5 (N5) RMT1 (R1) (Assuming RMT1 is on your IBM node 5). The 'B' in '(B,,CMD)' is the class of the card punch (at our site). The 'CMD' is a 'Form-ID'. This is a reserved Form-ID on the HP. It tells the HP to :STREAM this card deck.

You can also send files to the HP without using an HP job. This method uses Form-IDs (other than CMD) defined by you on the HP. This method allows anyone who knows the Form-ID to send a file to the HP and overwrite whatever is there with no security checking. As always, see the manual for more info.

You may have noticed that the previous example did not send data to the HP, just a command to be executed. You could easily replace the :TELLOP with :FCOPY FROM;TO=TOFILE;NEW followed by the data and :EOD. To make it more flexible, however, it is desirable to separate the HP job file from the IBM data file (both files reside on the IBM). For example:

```
//INSTREAM JOB ...
//*
//JS010    EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//* N5 not needed in DEST if RMT1 is on this node.
//SYSUT2   DD SYSOUT=(B,,CMD),DEST=R1
//SYSUT1   DD DSN=HPJCL
//         DD DSN=IBMDATA
//         DD DSN=HPEODEOJ
```

The last three lines are all part of the SYSUT1 DD statement. They cause all three data-sets (files) to be concatenated. Please note that all three should have the same LRECL (Logical RECord Length. Same as ;REC=), 80 bytes in this case.

The file HPJCL is:

```
!JOB INSTREAM,User/ABCD1234.Account;OUTCLASS=LP,1
!
!BUILD TOFILE;REC=-80,160,F,ASCII
!FCOPY FROM;TO=TOFILE
```

The file IBMDATA is:

```
Data record 1
Data record 2
Data record 3
Data record 4
```

The file HPEODEOJ is:

```
:EOD
!EOJ
```

Concatenated together, they create a valid HP job.

Simplifying the 'Simple' Transfer

Writing a lot of IBM JCL for each file I want to send is not my idea of a good time. Nor would I think that HP JCL would be fun for an IBM programmer. Wouldn't it be nice to keep the HP JCL on the HP and the IBM JCL on the IBM and send just the data (plus enough info to describe the target User and file name)? Well, we can get part way there. Let's start with the Reverse NRJE example this time.

On the IBM, the JCL would be:

```
//EXAMPLE1 JOB ...
//*
//JS010     EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN     DD DUMMY
//SYSUT2    DD SYSOUT=(B,,CMD),DEST=RMT1  << Same as R1 >>
//SYSUT1    DD DSN=HPJCL1
//          DD DSN=IBMDATA1
//          DD DSN=HPEOD1
```

The file HPJCL1 contains only the job card:

```
!JOB EXAMPLE1,NRJERECV/USERPW1.Account;OUTCLASS=LP,1
```

The file IBMDATA1 contains the familiar 4 data records.

The file HPEOD1 contains only an EOD card:

```
:EOD
```

The concatenated file (card deck) that is 'punched' to the HP and then :STREAMed would be:

```
!JOB EXAMPLE1,NRJERECV/USERPW1.Account;OUTCLASS=LP,1
Data record 1
Data record 2
Data record 3
Data record 4
:EOD
```

Now, to the casual observer, this would not appear to be a valid HP job. But I say, look again.

Of course, if we stream this job 'as is', it will flush as soon as the second record is read by MPE, but, MPE doesn't ever see it. We create a special User (NRJERECV) for each account that uses NRJE. This User is only for receiving files from NRJE. It is created with:

```
:NEWUSER NRJERECV.Account;CAP=AL,SF,ND,IA,BA;HOME=NRJERECV
```

The home group NRJERECV gives us a place to keep the incoming data if it needs extra processing on the way in. This User has a LOGON UDC set for it. The UDC runs MPEX and calls the command file NRJERECV.MPEXCMD.SYS (used by all accounts). This command file then calls the command file that has the same filename as the Job/Session ID for this job (The Job/Session ID is EXAMPLE1 so the command file is EXAMPLE1.NRJEPROC.Account). We keep NRJE job specific MPEX command files in the group NRJEPROC for the account that uses it. The UDC is:

```
USERLOGON
OPTION LOGON,NOBREAK
COMMENT  FILE=NRJERECV.UDC.Account
CONTINUE
RUN MPEX.PUB.VESOFT;INFO='XEQ NRJERECV.MPEXCMD.SYS'
CONTINUE
BYE
EOJ
*
```

The BYE in the above UDC ensures we are logged off whether in session or job mode. The UDC could have called the job specific command file directly. We prefer having all applications call the system wide command file first. This allows us to easily insert any processing that should be done for all inbound files. The system wide command file is:

```
:COMMENT FILE=NRJERECV.MPEXCMD.SYS
:COMMENT  Do processing common to all inbound files here.
:COMMENT  XEQ DOSTUFF
:COMMENT  Now, call job specific command file.
 XEQ !HPJOBNAME.NRJEPROC
```

Now for where the real work is done (the job specific command file):

```
:COMMENT FILE=EXAMPLE1.NRJEPROC.Account
:PURGE TOFILE
:BUILD TOFILE;REC=-80,160,F,ASCII
:FCOPY FROM;TO=TOFILE
:COMMENT  NOW DO ANY OTHER STUFF THIS JOB SHOULD DO.
```

Once the data is copied and the command file finishes, the UDC causes the HP job to log off.

ASCII to EBCDIC Translation

For the uninitiated, HPs encode characters in ASCII (American Standard Code for Information Interchange) while IBMs (mainframes not PCs) use EBCDIC (Extended Binary Coded Decimal Interchange Code). Appendix C of the FCOPY manual shows the conversion tables for these codes.

When NRJE receives a file from the IBM, the file is translated from EBCDIC to ASCII. There is no way to prevent this. Why would we want to? What data is not in either ASCII or EBCDIC?

Non-character (Binary) Data!

Think about it. Binary is binary, no encoding involved. What kind of data is in binary form? COBOL COMP and COMP-3 (packed decimal) fields (and their equivalents in whatever language).

So what we want is to be able to convert all of the DISPLAY type data (this can be numeric chars) and exclude the COMP and COMP-3 (binary) fields. But IBM's IEBGENER (pronounced FCOPY) has no way to do this, nor can you tell NRJE to not translate (Don't bother to call the Response Center, I already asked).

The answer, ugly as it is, is to accept the whole file from NRJE translated (note that the binary fields will look 'funny'). Then use FCOPY to translate the whole file back from ASCII to EBCDIC. Then use FCOPY (AGAIN!) to translate the file from EBCDIC to ASCII excluding the binary fields.

The last (job specific) command file would then be:

```
:COMMENT FILE=EXAMPLE1.NRJEPROC.Account
:PURGE TOFILE.NRJERECV  << Special group for extra inbound file processing >>
:BUILD TOFILE.NRJERECV;REC=-80,160,F,ASCII
:FCOPY FROM;TO=TOFILE;EBCDICOUT  << Translates ASCII to EBCDIC >>

:PURGE TOFILE.PUB
:BUILD TOFILE.PUB;REC=-80,160,F,ASCII
:FCOPY FROM=TOFILE.NRJERECV;TO=TOFILE.PUB;EBCDICIN=(1,4;10,4),EXCLUDE
```

The EBCDICIN keyword translates EBCDIC to ASCII. The EXCLUDE sub-parameter says not to translate the fields specified in the preceding list (Byte 1 for 4 bytes and byte 10 for 4 bytes). Note that FCOPY also has an INCLUDE sub-parameter. This causes only the specified fields to be translated. Use whichever one yields a smaller list. Also, if you ever change the record format for your data, the EXCLUDE/INCLUDE specs may also need to change.

For outbound files, you have more control over the translation. The NRJE Submit command has a Notranslate sub-parameter (it can be specified for any of the 5 submit files). To have NRJE translate your JCL but none of your data (assume it is all binary data):

```
RMT1> Submit  IBMJCL, HPDATA (Notranslate)
```

This gets the job done but how many of your files consist entirely of binary data? If you have a mixture of display and binary fields, you still need to translate the display fields while excluding the binary ones. So, just as in the inbound example, you FCOPY your data file, only this time using EBCDICOUT, excluding the binary fields. Since the display fields are now in EBCDIC, you may Submit your job with the Notranslate option on the datafile.

When using the standard translation table, not all ASCII characters translate to the same symbol in EBCDIC. The most notable are ! [ ] { }. There are a few others (See the paper *ASCII or EBCDIC: Only Your Printer Knows for Sure*, in these proceedings, for more specifics). When your HP job file comes in from the IBM, every '!' becomes a ']'. This is important if you choose to :STREAM the file yourself. You must `:STREAM file,]`. NRJE does this for you when you use the 'CMD' Form-ID.

Simpler HP to IBM Transfer

Since my IBM knowledge is rather limited, it was far easier to simplify the HP side than the IBM side. Also, my IBM counterparts tell me that JES does not have an equivalent for LOGON UDCs. This makes some sense since JES can run a job that has no User ID or Password specified as long as you don't want access to any disc files (datasets). By the way, DISC on an IBM is spelled DISK but more commonly known as DASD (Direct Access Storage Device). I tell you all of these wonderful IBM acronyms because you must learn their language. As they see it, you are trying to connect to their system. They are not trying to connect to yours.

The 'simplification' for the IBM consists of calling a PROC (PROCedure. A JCL sub-routine) and passing it the parameters for our specific file. We have 3 procs. One for records with 80 to 248 bytes. One for records with less than 80 bytes. And one for records with greater than 248 bytes (does reblocking). The procs are listed in the Appendix. In general, the procs purge the old file, copy the instream data to a new file and keep it as permanent. The proc that does reblocking has an extra file purge and file copy. So, on the HP you send the data by:

```
!NRJE RMT1
RMT1> Submit IBMJCL2, HPDATA80
RMT1> Exit
!
```

The file IBMJCL2 is:

```
//IBMJCL2  JOB (,B99),'HP to IBM test',CLASS=A,MSGCLASS=Z,
//         USER=NRJERCV,PASSWORD=WXYZ9876
//*
//* The User NRJERCV is used by everyone.  It is given write access only to
//* datasets that start with any High-level qualifier followed by NRJERECV
//* followed by anything (i.e.  Qualif.NRJERECV.*)
//*
//JS010    EXEC NRJERCV,
//         VDSN='XXEJV.NRJERECV.HPDATA80'
//* The DD card below overrides the dummy card in the proc.
//STEP020.SYSUT1  DD  DATA
```

The dataset name is substituted in the proc. The file HPDATA80 is any data you wish to send with REC=80.

Not so Simple Transfers

As you will soon see, 'simple' is a relative term. It can get much worse for record lengths other than 80 bytes. But record lengths come in all sizes so we need a way to send them somehow (easy or not).

NRJE has an upper limit on the record size. Inbound it is 255 bytes. Outbound is either 248 or 252 depending on the config. At our site it is 248.

For record sizes greater than 80, you must tell NRJE. This is done with the Submit command's Maxrec keyword.

```
RMT1> Submit IBMJCL3, HPDAT248; MAXREC=248
```

You must also alter the IBM JCL to match.

```
//IBMJCL3   JOB...
//JS010     EXEC NRJERCV
//          VDSN='XXEJV.NRJERECV.HPDAT248',
//          VLRECL=248,VBLKSIZ=2480
//STEP020.SYSUT1  DD  DATA
```

For record lengths less than 80, it gets a little tougher.

```
RMT1> Submit IBMJCL4, HPDATA40, LRECL40
```

The file IBMJCL4:

```
//IBMJCL4   JOB...
//JS010     EXEC NRJERCVL   << Proc for less than 80 bytes >>
//          VDSN='XXEJV.NRJERECV.HPDATA40',
//          VLRECL=40,VBLKSIZ=400
//STEP020.SYSUT1  DD  DATA
```

The file LRECL40 is appended after the instream data. It is:

```
/*      << This is the EOD for the instream data >>
//* File=LRECL40.NRJEPROC
//*
//* Override SYSIN  DD DUMMY in PROCstep 3
//*
//STEP030.SYSIN   DD  *
    GENERATE  MAXFLDS=1
    RECORD    FIELD=(40)
/*
```

This file adds instream IEBGENER commands. Note the 40 in the last command line matches our actual record length (REC=40).

When sending files from the IBM to the HP, you don't need to do near as much. Just make sure that all three of your files (JCL, DATA, EOD) have the same LRECL as the data file (80, 248, 40, or whatever).

This is all well and good but how do we send files larger than the upper physical limit? You can't with the file as is, but if you break each record in equal pieces that are less than 248, you can. The file must then be re-assembled on the other side (This is about the time you may begin to think that tapes aren't so bad after all).

File Surgery

Much to my surprise, after a few days trying, I found that once again FCOPY comes to the rescue. Say we want to send a 320 byte file. We do this by breaking each logical record in 2. We will end up with a file with twice as many 160 byte records (records may be broken in as many equal pieces as needed).

The original file was built with:

```
:BUILD DATA320;REC=-320,100,F,ASCII
```

The reblocked file is built with:

```
:BUILD RBLK160;REC=-160,200,F,ASCII
```

Note that the record length is halved while the blockfactor is doubled. This yields the same size block (physical record) for each file. We now tell FCOPY to copy blocks instead of logical records.

```
:FILE DATA320;MR;NOBUF
:FILE RBLK160;MR;NOBUF
:FCOPY FROM=*DATA320;TO=*RBLK160
```

You must admit, that's pretty easy. To send the file:

```
RMT1> Submit IBMJCL5, RBLK160; M=160
```

The file IBMJCL5:

```
//IBMJCL5  JOB ...
//*
//JS010    EXEC  NRJERCVR,   << Proc for greater than 248 bytes >>
//         VDSN='XXEJV.NRJERECV.HPDAT320',
//         VLRECL=320,VLRECLR=160,VBLKSIZ=3200
//*
//STEP020.SYSUT1  DD  DATA
```

Note that VLRECL is the final desired record length and VLRECLR is the reblocked record length.

Hardcode vs Defaults

When first implementing something like this, the temptation to hardcode things just to get it working can be great. And why not, it simplifies your testing. But once the testing is over, you need a method that is maintainable. Rest assured, something will change and the fewer changes you need to make because of it, the better. To insulate your HP jobs from the network you may wish to consider the following suggestions.

Assign each of your HPs a unique number. Then specify this number in your System level LOGON UDC with a Setjcw command. If the node number you assigned is 4 then use something like:

```
:SETJCW  HPSYSNUM = 4
```

This allows you to easily determine the node you are on from UDCs and Jobs with a simple IF command.

Now create a UDC at the System level called NRJE. This will override the MPE NRJE command. Have all Jobs and Sessions use the UDC to access NRJE (don't have jobs RUN NRJE.NRJE.SYS explicitly). The UDC will select the default WSID based on what node this is. This allows you to use the same UDC file on all systems without modification.

```
NRJE  WSID=" "
SETJCW  UDCPARM1BLANK      = 0
SETJCW  UDCPARM1BLANK!WSID = 1

IF   UDCPARM1BLANK = 0  THEN
     RUN NRJE.NRJE.SYS;INFO='!WSID'
ELSE
     IF   HPSYSNUM = 1  THEN
          RUN NRJE.NRJE.SYS;INFO='RMT1'
     ELSE
     IF   HPSYSNUM = 2  THEN
          RUN NRJE.NRJE.SYS;INFO='RMT2'
     ENDIF
     ENDIF
ENDIF
*
```

UDCs were used, rather than MPEX command files, on the Classic for speed reasons. Command files on MPE/iX should prove simpler and easier.

Now your jobs don't need to know the WSID to send a file. Just take the default as follows:

```
!NRJE
RMT1> Submit ...
RMT1> Exit
!
```

These defaults work fine when you have only a few WSIDs defined on the HP. At the IBM end, though, all of the WSIDs for all of your HPs are defined. Also, WSIDs are of the form RMTnnnn (nnnn= 1 to 9999). You must tell your IBM job which WSID to send its output to (this is essentially hardcoding the target HP node).

If your network has only one NRJE link, there is a way (quite complex) to send and receive files to and from any HP and IBM. It can also be somewhat slow but it does get there (and all of your systems can use NRJE for the price of one link).

Virtual NRJE

HP has a product, called SNA Server, that allows any of your HPs to send files through an NRJE link connected to only one of your HPs. As I understand it (I may be wrong), you cannot use Reverse NRJE with SNA Server. I have never used SNA Server so you may wish to investigate it. Assuming we don't want to spend any more money let's do it ourself.

Please take a moment to get some caffeine in your favorite form. This method uses many command files, utilities, and kludges and studying it will surely cure anyone's insomnia.

This method uses the concept of an 'NRJE hub' cpu (the one with the physical link). All NRJE traffic, inbound and outbound, passes through this node. Each direction is handled very differently and will be discussed separately. All of the jobs and command files are in the Appendix.

Virtual NRJE Inbound

Inbound means files sent from any IBM to any HP. Your IBM job needs a way to specify which HP node is the target (the WSID will only get the file as far as the NRJE hub node). We use the Form-ID and the NRJE lookup table to do this. The lookup table is where you define Form-IDs. The entries are very similar to file equates (I suspect that the entry is used verbatim in an actual file equate). In order to keep applications independent of each other and the node they reside on (you may move them someday) I suggest multiple Form-IDs for each node. At least one Form-ID per application per node. The file NRJETABL.NRJE.SYS might be:

```
HP1     =*HP01  << For System Mgmt use only >>
HP2     =*HP02          ''
HP10    =*HP10          ''
HP11    =*HP11          ''
AP      =*HP01  << Production system >>
GL      =*HP02          ''
APDV    =*HP10  << Development system >>
GLDV    =*HP10          ''
```

These are back references to file equates in the NRJE monitor job (streamed by the NRJECONTROL START command). The value on the left matches with the Form-ID in your IBM job.

```
//TESTAP   JOB ...
//*
//JS010 ...
//SYSUT2   DD SYSOUT=(B,,AP);DEST=N5R1  << DEST=N5R1 is in all jobs >>
```

OR

```
//TESTGL   JOB ...
//*
//JS010 ...
//SYSUT2   DD SYSOUT=(B,,GL);DEST=N5R1  << NOTE: Form-ID is GL not AP >>
```

When NRJE receives a file with Form-ID of GL, it routes it to the file equated with *HP02. In your NRJE monitor job, add the file equates:

```
:FILE HP01=HP01.NRJEJOB,OLD;EXC;ACC=APPEND
:FILE HP02=HP02...
```

The group NRJEJOB.SYS is where we keep all of the files related to our home grown, bidirectional SNA Server. The files HP## are message files built with:

```
:BUILD HP01.NRJEJOB;MSG;REC=-255,1,F;DISC=200000,32
```

Each message file is actually a queue for all inbound jobs for a specific node. The jobs (with their instream data) will be appended in this file one after another. You then need a monitor job (1 per target cpu) that waits on its message file. When data is received, the job starts copying the message file to a flat file until the message file is empty. The monitor job then logs on to its remote node, moves the flat file with DSCOPY, and :STREAMS the flat file on the remote node. The job then purges the flat files, or renames them, and waits on the message file again. When your data is tens of thousands of records, this method can get slow due to the many file copies.

If you use the lookup file to send inbound data/jobs to disc files, you may see an extra record (a blank one) at the end of each of your files. The Response Center tells me that this cannot be prevented. Why is the record there? Remember that the IBM is 'punching' cards to an RJE station (HP cpu). That last record causes the final 'card' to be ejected from the 'hopper'. Now that's what I call fanatical emulation! If you use jobs with instream data (rather than pure data) transfers, this extra record doesn't interfere as it shows up after the :EOD (or !EOJ).

Virtual NRJE Outbound

Here is another place where having the NRJE UDC can pay off. The UDC relies upon JCWs being set at logon. We have multiple cpus at our site. We prefer all of our UDCs and command files to be identical on all cpus so we can maintain just one master set on one cpu. All cpu specific files are in the group AACONFIG.SYS. One of these files is SYSJCWS. This file contains a list of :SETJCW commands with EXIT as the last record. This file is therefore valid as $STDIN for FCOPY. The SYSTEM LOGON UDC does a

```
:RUN FCOPY.PUB.SYS;STDIN=SYSJCWS.AACONFIG.SYS
```

This sets a whole list of JCWS for each job/session thus specifying the environment. A partial list is:

```
:SETJCW  HPSYSNUM = 4
:SETJCW  SYSCLASSNRJE = 0   << 0=Virtual.  1=Actual NRJE >>
:SETJCW  SYSNRJESUBMITGEN = 1  << Init Submit generation counter >>
EXIT
```

We now enhance our NRJE UDC to be:

```
NRJE  WSID=" "
SETJCW  UDCPARM1BLANK      = 0
SETJCW  UDCPARM1BLANK!WSID = 1

IF   SYSCLASSNRJE = 0  THEN
     COMMENT  USE VIRTUAL NRJE
     RUN MPEX.PUB.VESOFT;INFO='XEQ NRJE.MPEXCMD.SYS'
ELSE
     COMMENT  USE ACTUAL NRJE
     IF   UDCPARM1BLANK = 0  THEN
          RUN NRJE.NRJE.SYS;INFO='!WSID'
     ELSE
          IF   HPSYSNUM = 1  THEN
               RUN NRJE.NRJE.SYS;INFO='RMT1'
          ELSE
          IF   HPSYSNUM = 2  THEN
               RUN NRJE.NRJE.SYS;INFO='RMT2'
          ELSE
               DISPLAY "UNDEFINED HPSYSNUM (UDC NRJE)"
          ENDIF
          ENDIF
     ENDIF
ENDIF
*
DISPLAY MSG=" "
OPTION LIST
COMMENT !MSG
*
```

With this method, all jobs are independent of the physical network. If this node ever gets its own NRJE link, you simply change the SYSCLASSNRJE JCW from 0 to 1 and specify the WSID default in the UDC. Application jobs will not need to change at all.

The MPEX command file NRJE.MPEXCMD.SYS is the entry into the Virtual NRJE process. It calls the command file NRJEPARS which parses the Submit command.

These command files emulate most of the NRJE Submit command. The ##FD command is not emulated. ##FD acts like a compiler $INCLUDE command and is used from within any of the five Submit files. The emulation is done by parsing the Submit command, and encoding the parameters as :SETVARs in the file F0. The files to be Submitted (1 to 5) are copied to the files F1 through F5. Since we have a third party network spooling utility; and in order to make application jobs independent of any other cpu, the files F0 through F5 are actually spoolfiles. This allows a job to do its Submit and then log off. The NRJE hub cpu does not even need to be UP at this point. If and/or when the NRJE hub cpu is UP, the network spooler copies the F0 through F5 files to the hub. On the NRJE hub cpu is a 'monitor' job. This job scans for all spoolfiles for outbound NRJE. When it finds some, it copies them to disc files, decodes the parms and does a Submit using real NRJE.

I have left out some very important details in the above discussion. The monitor job on the NRJE hub needs to know several things to be able to properly reconstruct the original Submit from the spoolfiles. It needs to know:

- What is the source node?
- Which file is F0, F1, etc...?
- Has the whole group (F0 through Fn) made it or are some still in transit?
- What job/session on the source node sent this group?
- Which group within a job/session is this (a job/session could do many Submits one right after another)? I refer to each group as a Submit generation.

The first kludge is to define a group of class names on the hub. One group per source node. One class per Fn file. For node 1 we would need:

```
NRJE01F0
NRJE01F1
NRJE01F2
NRJE01F3
NRJE01F4
NRJE01F5
```

For node 22 we would add the classes:

```
NRJE22F0
NRJE22F1
NRJE22F2
NRJE22F3
NRJE22F4
NRJE22F5
```

As you can see, this could amount to a lot of class names. These class names are the target of a set of class names on each source node. The source class to target class is defined in the network spooler. The source classes are:

```
NRJEF0
NRJEF1
NRJEF2
NRJEF3
NRJEF4
NRJEF5
```

These classes are the same on all source nodes. With the use of class names, we have satisfied the questions:

- Which source node?
- Which Fn file?

When the source files are written to the spooler, the filename is actually the job/session type and number, the number of Fn files and which generation for this job/session. The filename is of the form: ?nnnnngg

> '?' is the leading letter.
>
> 'nnnnn' is the job/session number.
>
> 'gg' is the Submit generation (how many Submits for this job/session).

The first letter of the file name can be (H,I,J,K,L) for jobs or (Q,R,S,T,U) for sessions. If there is only an F1 file (plus the F0 which is not counted), then the file will start with either 'H' for a job or 'Q' for a session. These letters were chosen because we usually Submit 3 files (IBMJCL, HPDATA, MOREJCL). This would cause the 'J' and 'S' to be used most of the time at our site. (The file MOREJCL is some IBM JCL to tell the IBM job scheduler that our foreign job has finished. This satisfies the dependency for the next job in the IBM production run. Just ignore this if it doesn't make sense).

Using this naming convention, we have now answered the questions:

- Which job/session?
- Which Submit generation?

When the monitor job finds a ready file on any of the NRJEnnF0 classes, it determines how many files are in the group from the first letter of the filename. It then looks at the NRJEnnF1 through NRJEnnF5 classes for files with a matching filename. It must find the proper quantity and all in a READY status or the group is skipped on this pass.

If a group has totally completed transferring from the source node, the monitor then copies the group, each spoofle to a separate disc file, decodes the parms in the F0 file, calls NRJE and Submits the disc files. The monitor then goes on to the next group.

This process can put a heavy load on your cpu. We had a dedicated S/70 for our network hub so it wasn't a problem.

HP to IBM to Data General and Back

At our site, we have DGs (Data Generals) in addition to HPs and IBMs. The DGs also have an RJE facility to the IBMs. Our HPs trade tapes directly with the DGs but since the DGs have RJE, why not use it.

I know even less about DGs than IBMs so the DG explanation may be a little weak. This example shows the power of RJE to connect all of your cpus. We are going to show how a single job file on the HP can be used to test the network from HP to IBM to DG back to IBM and back to the originating HP. It is a simple matter of embedding an HP job within an IBM job within a DG job within an IBM job within an HP job (got all that?). Such as:

```
HP [IBM [DG [IBM [HP]]]]
```

From the point of view of each job, everything to the right of the '[' is data, but is actually a job for the cpu that receives it.

```
#JOB TESTNRJE,MANAGER.SYS;OUTCLASS=LP,1
#
#COMMENT  THIS JOB MUST BE STREAMED WITH :STREAM TESTNRJE,#
#COMMENT  THIS IS SO THE EMBEDDED HP JOB WILL BE SEEN AS DATA.
#
#NRJE
SUBMIT $STDINX
//HPIBMDG JOB (,B24),'HP-IBM-DG-IBM-HP',CLASS=A,MSGCLASS=Z
//*
//* NOTE: DEST=N27R18 is the node (27) and RMT (18) of the line
//*       that is attached to the DG
//*
//COPY     EXEC  PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//* The punch is class Y on node 27.
//SYSUT2   DD SYSOUT=(Y),
//            DCB=(BLKSIZE=80,LRECL=80,RECFM=U),DEST=N27R18
// Specify a substitute EOD indicator as '*/'.
//SYSUT1   DD DATA,DLM='*/'
//*
//* ##FD is needed for the Transparent parm.  This allows the
//* second IBM job to be passed through as data rather than processed
//* immediately as another job in this input stream.
##FD $STDINX (T)
$$JOB OP    << DG job card >>
$$PASSWORD AB12DC34
DEL/2=IGNORE :UDD:VISTICA:SEND.TMP    << Purge file >>
CRE/I :UDD:VISTICA:SEND.TMP    << Build file. Load with instream data >>
//HPIBMDG  JOB (,B99),'ERIK VISTICA',CLASS=A,MSGCLASS=Z
//*
//* Send this job to a different IBM node for execution.
/*ROUTE XEQ XXXMVS1
/*ROUTE PRINT XXXMVS1
//*
```

```
//*
//COPY     EXEC  PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT2   DD SYSOUT=(B,,CMD),DEST=N1R101
//SYSUT1   DD *
!JOB NRJEOK,MANAGER.SYS;OUTCLASS=,1
!TELL MANAGER.SYS HP/IBM/DG/IBM/HP TEST OK.
!SET STDLIST = DELETE
!EOJ
)    << DG EOD >>
CRJ SEND :UDD:VISTICA:SEND.TMP    << Access DG RJE subsystem >>
BYE
$$END    << End of DG job >>
*/       << Substitute EOD for IBM >>
:EOD     << For ##FD $STDINX >>
:EOD     << For SUBMIT $STDINX >>
EXIT     << Exit NRJE subsystem >>
#
#EOJ
```

You could use this same method if you had a string of cpus alternating HP and IBM that you need to get through.

In order to do RJE with any vendor's cpu, you need to determine their word(s) for the following concepts:

- JOB card
- FILE equate
- FCOPY program
- EOD card
- EOJ card
- STREAM command
- Access to the RJE line. For HP this is the NRJE Submit command. On the IBM this is the //SYSUT2 DD SYSOUT=(B,,CMD),DEST=N1R1.

Security

By the very nature of NRJE (a job as a card deck) you need to embed the password in the JOB card. With vanilla NRJE you must hardcode this. The concept of a shared user with limited capability makes this restriction somewhat easier to live with. You could possibly have a program that inserts the password for you (much like STREAMX does) but getting it to read the password from the IBM could be difficult. Anyway, IBM security types don't like this kind of hole. Depends on whether you view your systems as HP vs IBM or as a network of processors that happen to run different operating systems. If you view them as the latter, then some kind of security interface program makes a lot of sense. Alternatively, you could keep a copy of only the needed passwords for the HP on the IBM and vise versa (but who wants to maintain it). As you can see, the effort to implement security increases rapidly if you go beyond the single function, shared user method. Any ideas?

Other Tidbits

When the file is transferred either to or from the IBM, I think that it always has a record length of 255 bytes, a blocking factor of 1, and Undefined record format.

There is a config option for CKPTLNS (checkpoint lines) and CKPTPGS (checkpoint pages). This is also called Checkpoint Restart. It can be used to establish a 'checkpoint' every so often during the transfer. Then if the link fails, only the portion of the file past the last checkpoint needs to be sent when the link comes back UP. We have set both CKPTLNS and CKPTPGS to their maximum values. This sets a checkpoint so large that we will never reach it, thus effectively disabling the feature. The reason is because we don't send pure data files but rather Jobs with instream data. We had a monthly problem with IBMs at one end of the country sending data to HPs at the other end. Once a month during the night (usually in the middle of the transfer), the network (managed by a third site) would go down, without warning, for maintenance. So, the JOB card would reach the HP with some of the data. The network would be brought down, by the third site, for maintenance. The HP would think everything was fine. The network would come back up. Checkpoint Restart would cause the rest of the data plus the :EOD to be sent. The HP would simply discard this data because no JOB card was found. In all fairness, I expect that Checkpoint Restart can be very useful. We just did not need it enough to pursue making it work for us.

If both your HPs and IBMs are in the same room, you may be tempted to run a cable straight from the INP to the IBM 37xx controller. I tried and was unable to create a suitable cable. HP tells me that the clock on the INP is not good enough to drive the line and they don't know if the IBM controller can. A pair of MODEMs wired back to back works great. It also makes the two systems independent of each other. When the IBM IPLs (Initial Program Load - a Coldload), the Line (link), the PU (Physical Unit), and LUs (Logical Units) will attempt to go active. If the MODEM is powered up then the link will stay in a PCT2 (Pending Contact 2) state until SNA is started on the HP. If you were hardwired and the link was not started on the HP, I suspect that the IBM would give up on the spot. You would then have to manually start the line. Besides, the MODEM LEDs help a great deal when troubleshooting link problems.

Remember that the NRJE Submit command copies your file to a spoolfile before it is transferred. If you transfer very large files, you may need to increase the value for '# OF SECTORS PER SPOOLFILE EXTENT'. This is under 'DISC ALLOCATION CHANGES' in SYSDUMP.

Conclusion

NRJE can be an asset to your data center. The investment you make in learning to use it pays off with a stable, automated interface. I hope that this information will enable you to gain more utility from NRJE. Tapes and NRJE, use the best of each world.

APPENDIX

Disclaimer

These examples have been used successfully by ROLM. However, NO warranty is made to the accuracy of the examples or any related material, and NO responsibility is assumed for any modification directly or indirectly caused by the use of these examples. It is the user's responsibility to evaluate the usefulness of the material supplied. Also, NO guarantee is made to provide any corrections or extensions made in the future.

HP to IBM Examples

For 80 byte records:

Within your HP job:

```
!NRJE
SUBMIT  IBMJOB80, HPDATA80
EXIT
!
```

The file IBMJOB80:

```
//XXEJV80  JOB (,B24),'HP to IBM 80b',CLASS=A,MSGCLASS=Z,
//         USER=XXEJV,PASSWORD=ABCD1234
//*
//JS010    EXEC NRJERCV,
//         VDSN='XXEJV.NRJERECV.TST80'
//STEP020.SYSUT1  DD DATA
```

For 80 to 248 byte records
(248 in this case)

Within your HP job:

```
!NRJE
SUBMIT  IBMJB248, HPDAT248; MAXREC=248
EXIT
!
```

The file IBMJB248:

```
//XXEJV248 JOB (,B24),'HP to IBM 248b',CLASS=A,MSGCLASS=Z,
//         USER=XXEJV,PASSWORD=ABCD1234
//*
//JS010    EXEC NRJERCV,
//         VDSN='XXEJV.NRJERECV.TST248',
//         VLRECL=248,VBLKSIZ=2480
//STEP020.SYSUT1  DD DATA
```

For less than 80 byte records
(40 in this case)

Within your HP job:

```
!NRJE
SUBMIT  IBMJOB40, HPDATA40, LRECL40
EXIT
!
```

The file IBMJOB40:

```
//XXEJV40  JOB (,B24),'HP to IBM 40b',CLASS=A,MSGCLASS=Z,
//         USER=XXEJV,PASSWORD=ABCD1234
//*
//JS010    EXEC NRJERCVL,
//         VDSN='XXEJV.NRJERECV.TST40',
//         VLRECL=40,VBLKSIZ=400
//STEP020.SYSUT1  DD DATA
```

The file LRECL40:

```
/*
//* File=LRECL40.NRJEPROC
//* The first record of this file is the EOD for the INSTREAM data.
//* This file contains instream commands for IEBGENER.
//* Override DD
//STEP030.SYSIN    DD *
     GENERATE  MAXFLDS=1
     RECORD    FIELD=(40)
/*
```

For greater than 248 byte records
(320 in this case)

The original data file was built with:

```
!BUILD HPDT320;REC=-320,100,F,ASCII
```

Within your HP job:

```
!COMMENT  REBLOCK THE DATA.
!BUILD HPDT160R;REC=-160,200,F,ASCII
!FILE HPDT320 ;MR;NOBUF
!FILE HPDT160R;MR;NOBUF
!FCOPY FROM=*HPDT320;TO=*HPDT160R
!
!NRJE
SUBMIT  IBMJB320, HPDT160R; M=160
EXIT
!
```

The file IBMJB320:

```
//XXEJV320 JOB (,B24),'HP to IBM 320b',CLASS=A,MSGCLASS=Z,
//         USER=XXEJV,PASSWORD=ABCD1234
//*
//JS010    EXEC NRJERCVR,
//         VDSN='XXEJV.NRJERECV.TST320',
//         VLRECL=320,VLRECLR=160,VBLKSIZ=3200
//STEP020.SYSUT1  DD DATA
```

IBM to HP Examples

For less than or equal to 248 byte records

(80 bytes in this case)

The DATA file has BLKSIZE = 8000 (The JOBCARD and EOD files actually have BLKSIZE = 80. All three files have LRECL = 80):

```
//XXEJV80  JOB (,B24),'IBM TO HP 80B',CLASS=A,MSGCLASS=Z,
//         USER=XXEJV,PASSWORD=ABCD1234
//COPY     EXEC  PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT1   DD DSN=XXEJV.NRJE.TST80.JOBCARD,DISP=OLD,
//            DCB=(BLKSIZE=8000)
//         DD DSN=XXEJV.NRJE.TST80.DATA,DISP=OLD
//         DD DSN=XXEJV.NRJE.TST80.EOD,DISP=OLD,
//            DCB=(BLKSIZE=8000)
//SYSUT2   DD SYSOUT=(B,,CMD),DEST=N1R101
```

For greater than 248 byte records

(320 in this case)

For the file IBMJB320 , the JOBCARD and EOD files have LRECL = 160 and BLKSIZE = 160. The DATA file actually has LRECL = 320 and BLKSIZE = 3200. Note that the LRECL = 160 is half the actual LRECL of 320. This causes the file SYSUT2 to have twice as many 160 byte records.

```
//XXEJV320 JOB (,B24),'IBM TO HP 320B',CLASS=A,MSGCLASS=Z,
//         USER=XXEJV,PASSWORD=ABCD1234
//COPY     EXEC  PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT1   DD DSN=XXEJV.NRJE.TST160.JOBCARD,DISP=OLD,
//            DCB=(BLKSIZE=3200)
//         DD DSN=XXEJV.NRJE.TST320.DATA,DISP=OLD,
//            DCB=(LRECL=160)
//         DD DSN=XXEJV.NRJE.TST160.EOD,DISP=OLD,
//            DCB=(BLKSIZE=3200)
//SYSUT2   DD SYSOUT=(B,,CMD),DEST=N1R101
```

Sample IBM Procs

For 80 to 248 byte records

```
//NRJERCV  PROC  VDSN='NRJERECV.DATA.DEFAULT',
//         VLRECL=80,VBLKSIZ=8000,
//         VSPACE='(TRK,(1,1),RLSE)',
//         VUNIT=PROD,
//         VSYSOUT='*'
//*********************************************************************
//* I COPY INSTREAM DATA TO DASD (REPLACES TAPE TRANSFERS).
//* USE FOR RECORDS WITH 80 TO 248 BYTES.
//* USE NRJERCVL FOR LESS THAN 80. USE NRJERCVR FOR MORE THAN 248.
//*
//* I AM CALLED BY JOBS SENDING DATA TO IBM FROM OTHER CPUS.
//* THE CALLING JOB CONSISTS OF A JOB CARD, AN EXEC CARD THAT CALLS
//* THIS PROC, A SYSUT1 DD DATA CARD FOR THE EXEC OF IEBGENER IN THIS
//* PROC, AND THE INSTREAM DATA.
//*
//*
//*DELETE DATASET
//STEP010  EXEC PGM=IEFBR14
//PERMDEL  DD  DSN=&VDSN,
//             DISP=(MOD,DELETE,DELETE),SPACE=(TRK,0)
//*
//*
//*COPY INSTREAM DATA TO PERM DATASET
//STEP020  EXEC PGM=IEBGENER
//SYSIN    DD  DUMMY
//SYSPRINT DD  SYSOUT=&VSYSOUT
//*OVERRIDE THE NEXT DD CARD WHEN CALLING THIS PROC.
//SYSUT1   DD  DUMMY
//SYSUT2   DD  DSN=&VDSN,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=&VSPACE,
//             UNIT=&VUNIT,
//             DCB=(RECFM=FB,LRECL=&VLRECL,BLKSIZE=&VBLKSIZ)
//************************* END  NRJERCV  *****************************
```

For less than 80 byte records

```
//NRJERCVL PROC  VDSN='NRJERECV.DATA.DEFAULT',
//         VLRECL=80,VBLKSIZ=8000,
//         VSPACE='(TRK,(1,1),RLSE)',
//         VUNIT=PROD,
//         VSYSOUT='*'
//*****************************************************************
//* I COPY INSTREAM DATA TO DASD (REPLACES TAPE TRANSFERS).
//* USE FOR RECORDS WITH LESS THAN 80 BYTES.
//* USE NRJERCV FOR 80 TO 248. USE NRJERCVR FOR MORE THAN 248.
//*
//* I AM CALLED BY JOBS SENDING DATA TO IBM FROM OTHER CPUS.
//* THE CALLING JOB CONSISTS OF A JOB CARD, AN EXEC CARD THAT CALLS
//* THIS PROC, A SYSUT1 DD DATA CARD FOR THE EXEC OF IEBGENER IN THIS
//* PROC, AND THE INSTREAM DATA.
//*
//*
//*DELETE DATASETS
//STEP010  EXEC PGM=IEFBR14
//REBLKDEL DD  DSN=&VDSN..REBLK,
//             DISP=(MOD,DELETE,DELETE),SPACE=(TRK,0)
//PERMDEL  DD  DSN=&VDSN,
//             DISP=(MOD,DELETE,DELETE),SPACE=(TRK,0)
//*
//*
//*COPY INSTREAM DATA TO REBLOCK DATASET
//STEP020  EXEC PGM=IEBGENER
//SYSIN    DD  DUMMY
//SYSPRINT DD  SYSOUT=&VSYSOUT
//*OVERRIDE THE NEXT DD CARD WHEN CALLING THIS PROC.
//SYSUT1   DD  DUMMY
//SYSUT2   DD  DSN=&VDSN..REBLK,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=&VSPACE,
//             UNIT=&VUNIT,
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//*
//*
//*COPY REBLOCK DATASET TO PERM DATASET
//STEP030  EXEC PGM=IEBGENER
//*OVERRIDE THE NEXT DD CARD WHEN CALLING THIS PROC.
//SYSIN    DD  DUMMY
//SYSPRINT DD  SYSOUT=&VSYSOUT
//SYSUT1   DD  DSN=&VDSN..REBLK,DISP=OLD
//SYSUT2   DD  DSN=&VDSN,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=&VSPACE,
//             UNIT=&VUNIT,
//             DCB=(RECFM=FB,LRECL=&VLRECL,BLKSIZE=&VBLKSIZ)
//*
```

```
//*
//*DELETE REBLOCK DATASET
//STEP040  EXEC PGM=IEFBR14,
//         COND=(0,NE)
//REBLKDEL DD  DSN=&VDSN..REBLK,
//             DISP=(MOD,DELETE,DELETE),SPACE=(TRK,0)
//************************** END  NRJERCVL  ****************************
```

For greater than 248 byte records

```
//NRJERCVR PROC  VDSN='NRJERECV.DATA.DEFAULT',
//         VLRECL=80,VLRECLR=80,VBLKSIZ=8000,
//         VSPACE='(TRK,(1,1),RLSE)',
//         VUNIT=PROD,
//         VSYSOUT='*'
//*********************************************************************
//* I COPY INSTREAM DATA TO DASD (REPLACES TAPE TRANSFERS).
//* USE FOR RECORDS WITH MORE THAN 248 BYTES.
//* USE NRJERCV FOR 80 TO 248 BYTES. USE NRJERCVL FOR LESS THAN 80.
//*
//* I AM CALLED BY JOBS SENDING DATA TO IBM FROM OTHER CPUS.
//* THE CALLING JOB CONSISTS OF A JOB CARD, AN EXEC CARD THAT CALLS
//* THIS PROC, A SYSUT1 DD DATA CARD FOR THE EXEC OF IEBGENER IN THIS
//* PROC, AND THE INSTREAM DATA.
//*
//* WHEN SUBMITTED FROM HP/NRJE, THE MAXREC VALUE MUST EQUAL VLRECLR.
//*
//*
//*DELETE DATASETS
//STEP010  EXEC PGM=IEFBR14
//REBLKDEL DD  DSN=&VDSN..REBLK,
//             DISP=(MOD,DELETE,DELETE),SPACE=(TRK,0)
//PERMDEL  DD  DSN=&VDSN,
//             DISP=(MOD,DELETE,DELETE),SPACE=(TRK,0)
//*
//*
//*COPY INSTREAM DATA TO REBLOCK DATASET
//STEP020  EXEC PGM=IEBGENER
//SYSIN    DD  DUMMY
//SYSPRINT DD  SYSOUT=&VSYSOUT
//*OVERRIDE THE NEXT DD CARD WHEN CALLING THIS PROC.
//SYSUT1   DD  DUMMY
//SYSUT2   DD  DSN=&VDSN..REBLK,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=&VSPACE,
//             UNIT=&VUNIT,
//             DCB=(RECFM=FB,LRECL=&VLRECLR,BLKSIZE=&VBLKSIZ)
//*
//*
//*COPY REBLOCK DATASET TO PERM DATASET
//STEP030  EXEC PGM=IEBGENER
//SYSIN    DD  DUMMY
//SYSPRINT DD  SYSOUT=&VSYSOUT
//SYSUT1   DD  DSN=&VDSN..REBLK,DISP=OLD,DCB=(LRECL=&VLRECL)
//SYSUT2   DD  DSN=&VDSN,
//             DISP=(NEW,CATLG,DELETE),
//             SPACE=&VSPACE,
//             UNIT=&VUNIT,
//             DCB=(RECFM=FB,LRECL=&VLRECL,BLKSIZE=&VBLKSIZ)
```

```
//*
//*
//*DELETE REBLOCK DATASET
//STEP040  EXEC PGM=IEFBR14,
//         COND=(0,NE)
//REBLKDEL DD  DSN=&VDSN..REBLK,
//             DISP=(MOD,DELETE,DELETE),SPACE=(TRK,0)
//************************* END  NRJERCVR  ****************************
```

HP to IBM Jobs and Command Files

```
!JOB JSUBMON,NRJEJOBQ.SYS,NRJE;OUTCLASS=,1;PRI=CS;INPRI=12
!
!COMMENT  I RUN ON THE NRJE HUB AND WATCH FOR SPOOL.DEV = NRJE##S0.
!COMMENT  WHEN I FIND ONE, I SUBMIT THE RELATED GROUP OF FILES TO
!COMMENT  NRJE.
!
!TELL LOGECHO.SYS JSUBMON - LOGON COMPLETED.  START MPEX...
!
!RUN MPEX.PUB.VESOFT
 :SETVAR  MPEXCMDTRACE = 0
 :SETVAR  HPAUTOCONT = TRUE

 :SETVAR  INFINITE = 1
  WHILE   INFINITE = 1
    XEQ JSUBMON.AACONFIG

    COMMENT  IF NRJRMON IS DOWN, THIS IS THE LAST TIME THRU.
    IF   JSCOUNT ('!WSID,NRJEMON.SYS') = 0  THEN
         SETVAR  INFINITE = 0
    ENDIF

    XEQ JSUBMAIN.NRJEJOB  !WSID

    IF   INFINITE = 1  THEN
 :       !TEL !HPDATEF !HPTIMEF  - JSUBMON      WAIT !W SECONDS.
         CALC PAUSE (!W)
    ENDIF
  ENDWHILE

EXIT
!
!TELL LOGECHO.SYS JSUBMON - LOGGING OFF...
!EOJ
```

```
:COMMENT  MPEX CMD FILENAME = JSUBMON.AACONFIG
:SETVAR  WSID = 'MVS1R101'
:SETVAR  SUBFENCE = 5
:SETVAR  W = 90
:SETVAR  TEL = 'TELL LOGECHO.SYS'
```

```
PARM  WSID
:COMMENT  MPEX CMD FILENAME = JSUBMAIN

:COMMENT  I RUN ONLY ON THE NRJE HUB NODE.  I TAKE FILES FROM THE
:COMMENT  SPOOLER AND SUBMIT THEM TO NRJE.

:COMMENT  SPOOFLES THAT COME FROM A JOB START WITH (H,I,J,K,L).
:COMMENT    ''    ''   ''   ''  A SES  ''    ''  (Q,R,S,T,U).
:COMMENT  THE LETTER TELLS US HOW MANY FILES ARE IN THE GROUP.
:COMMENT  ('H' AND 'Q' MEAN 1, 'L' AND 'U' MEAN 5).  SINCE THERE
:COMMENT  WILL USUALLY BE 3 FILES IN THE GROUP (JOB + DATA) YOU WILL
:COMMENT  SEE FILENAMES STARTING WITH 'J' OR 'S' MOST OFTEN.

:SETVAR  MPEXHINT = 0

:REPEAT
:  SETVAR  N0            = '![SPOOL.DEVICE [4:2]]'
:  SETVAR  SPOOL_FILE    = '![SPOOL.FILE]'
:  SETVAR  SPOOL_ACCOUNT = '![SPOOL.ACCOUNT]'
:  SETVAR  DEVGROUP      = 'NRJE' + '!N0' + 'S#'

:  COMMENT  MAKE SURE THE WHOLE GROUP HAS BEEN RECEIVED.
:  !TEL !HPDATEF !HPTIMEF  - &
JSUBMAIN      CHECK FOR COMPLETE GROUP.
:  COMMENT  DON'T COUNT THE NRJE##S0 FILE.
:  SETVAR  RQTY = -1
:  REPEAT
:    SETVAR  RQTY = RQTY + 1

:    COMMENT  PURGE ANY OLD, INCOMPLETE GROUPS.
!    SET DATE,YMD
:    IF   ![SPOOL.READYDATE] < TODAY - 1  THEN
:         !TEL !HPDATEF !HPTIMEF  - &
JSUBMAIN      PURGE SPOOFLE FROM OLD, INCOMPLETE GROUP.
:         DELETESPOOLFILE #O![SPOOL.SPOOLFILENUM]
:    ENDIF
!    SET DATE,MDY
   FORFILES  !SPOOL_FILE.@.!SPOOL_ACCOUNT&
           (SPOOL.DEVICE MATCHES '!DEVGROUP' AND SPOOL.ISREADY):SPOOL

:  SETVAR  FQTY = ORD ('!SPOOL_FILE[0:1]') - 64 - 7
:  IF   FQTY > 5  THEN
:       SETVAR  FQTY = FQTY - 9
:  ENDIF


:  COMMENT  PROCESS GROUP IF ALL HAS BEEN RECEIVED.
:  IF   RQTY <> FQTY  THEN
:       !TEL !HPDATEF !HPTIMEF  - &
JSUBMAIN      GROUP NOT COMPLETE !SPOOL_FILE.!DEVGROUP.
:  ELSE
:       COMMENT  MPEX SON NEEDED TO PREVENT STACK OVERFLOWS.
:       RUN MPEX.PUB.VESOFT;INFO="XEQ JSUBSPDF.NRJEJOB"

:       COMMENT  WE NOW HAVE THE GROUP OF SPOOFLES AS DISCFILES.
:       COMMENT  INSERT A ':SETVAR' IN EACH RECORD IN FILE F0.
:       !TEL !HPDATEF !HPTIMEF  - JSUBMAIN      INSERT SETVARS.
:       RUN EDITOR.PUB;&
          INPUT='T F0.NRJEJOB;CQ1,":SETVAR ",ALL;K F0.NRJEJOB,UNN;E'
        USEQ F0.NRJEJOB

:       !TEL !HPDATEF !HPTIMEF  - JSUBMAIN      SUBMIT THRU NRJE.
:       RUN NRJE.NRJE;INFO="!WSID";INPUT=&
          SUBMIT *F1 !FN1,*F2 !FN2,*F3 !FN3,*F4 !FN4,*F5 !FN5 !PARMS'
:  ENDIF

 FORFILES  @.@.@(SPOOL.DEVICE MATCHES 'NRJE##S0'&
            AND SPOOL.ISREADY  AND SPOOL.OUTPRI > !SUBFENCE):SPOOL

:COMMENT  END JSUBMAIN
```

```
:COMMENT  MPEX CMD FILENAME = JSUBSPDF

:COMMENT  BUILD NULLSRC.NRJEJOB;REC=-2,1,F,ASCII;DISC=1,1
:FILE F2=NULLSRC.NRJEJOB,OLD
:FILE F3=NULLSRC.NRJEJOB,OLD
:FILE F4=NULLSRC.NRJEJOB,OLD
:FILE F5=NULLSRC.NRJEJOB,OLD

:COMMENT  FIND ALL RELATED FILES (F0 THRU F5).
:SETVAR  DONE = FALSE
:SETVAR INDEX  = 0
:WHILE (INDEX <= 5) &
   AND NOT DONE

:  COMMENT  FOR EACH SPECIFIC F0-F5 FILE:
:  SETJCW  CIERROR = OK
:  REPEAT
:    COMMENT  MAKE DISCFILE FROM SPOOFLE.
:    !TEL !HPDATEF !HPTIMEF  - &
JSUBSPDF       SPOOFLE TO DISC !SPOOL_FILE.![SPOOL.DEVICE]
     NOMSG  :PURGE F!INDEX.NRJEJOB
:    SETJCW  CIERROR = OK
:    FILE DISCFILE=F!INDEX.NRJEJOB;DEV=DISC;DISC=100000,32;&
          REC=-248,1,V,ASCII;NOCCTL
:    RUN SPOOK5.PUB.SYS;INPUT='COPY ![SPOOL.SPOOLFILENUM];&
        ALL,*DISCFILE';INPUT='QUIT'

:    FILE F!INDEX=F!INDEX.NRJEJOB,OLD

:    DELETESPOOLFILE #0![SPOOL.SPOOLFILENUM]

   FORFILES  @.@.@ (SPOOL.DEVICE='NRJE!"NO"S!INDEX' &
        AND  SPOOL.ISREADY  AND SPOOL.FILE='!SPOOL_FILE'):SPOOL

:  IF   CIERROR <> OK  THEN
:       SETVAR  DONE = TRUE
:  ELSE
:       SETVAR  INDEX = INDEX + 1
:  ENDIF
 ENDWHILE

:COMMENT  END JSUBSPDF
```

IBM to HP Jobs and Command Files

```
!JOB JHPNNMON,NRJEJOBQ.SYS,NRJE;OUTCLASS=,1;PRI=CS;INPRI=12
!
!COMMENT  I AM STREAMED BY !WSID AND RUN ON THE NRJE HUB NODE.
!COMMENT  I WATCH THE HP##.NRJEJOB.SYS FILES WAITING FOR DATA.
!COMMENT  WHEN ONE OF THEM RECEIVES DATA, I STREAM A JOB TO FORWARD
!COMMENT  THAT DATA TO ITS TARGET NODE (IF NO JOB CURRENTLY EXISTS).
!COMMENT  A JOB IS NEEDED INSTEAD OF A SON PROCESS TO PREVENT TIMING
!COMMENT  CONFLICTS WHEN CHECKING THE JCW CIERROR.
!COMMENT  I ALSO WATCH THE JOB QUEUE LOOKING FOR !WSID,NRJEMON.SYS
!COMMENT  TO LOG OFF.
!
!TELL LOGECHO.SYS JHPNNMON - LOGON COMPLETED. START MPEX...
!
!CONTINUE
!RUN MPEX.PUB.VESOFT;INFO="XEQ JHPNMAIN.NRJEJOB"
!
!TELL LOGECHO.SYS JHPNNMON - LOGGING OFF...
!EOJ
```

```
:COMMENT  FILE = JHPNMAIN.NRJEJOB.SYS

:COMMENT  I AM CALLED FROM JHPNNMON.NRJEJOB.SYS
:COMMENT  I WATCH THE HP##.NRJEJOB.SYS FILES WAITING FOR DATA.
:COMMENT  WHEN ONE OF THEM RECEIVES DATA, I STREAM A JOB TO FORWARD
:COMMENT  THAT DATA TO ITS TARGET NODE (IF NO JOB CURRENTLY EXISTS).
:COMMENT  A JOB IS NEEDED INSTEAD OF A SON PROCESS TO PREVENT TIMING
:COMMENT  CONFLICTS WHEN CHECKING THE JCW CIERROR.
:COMMENT  I ALSO WATCH THE JOB QUEUE LOOKING FOR !WSID,NRJEMON.SYS
:COMMENT  TO LOG OFF.

:!TEL !HPDATEF !HPTIMEF  - JHPNNMON     MPEX ACTIVATED.

:SETVAR  MPEXCMDTRACE = 0
:SETVAR  HPAUTOCONT = TRUE

:COMMENT  WATCH THE JOB QUEUE FOR !WSID TO LOG OFF.  ALSO,
:COMMENT  START A JOB FOR EACH HP##.NRJEJOB.SYS FILE THE
:COMMENT  FIRST TIME IT HAS DATA.
:SETVAR  INFINITE = 1
 WHILE   INFINITE = 1
   XEQ JHPNNMON.AACONFIG

:  COMMENT  IF NRJRMON IS DOWN, THIS IS THE LAST TIME THRU.
:  IF   JSCOUNT ('!WSID,NRJEMON.SYS') = 0  THEN
:       SETVAR  INFINITE = 0
:       !TEL !HPDATEF !HPTIMEF  - JHPNNMON      !WSID HAS LOGGED OFF.
:  ELSE
```

```
        REPEAT
:         IF   RFILE.EOF > 0  THEN
:              !TEL !HPDATEF !HPTIMEF  - JHPNNMON      ![RFILE.FILE] H

               SETVAR  N = INTEGERPARSE (RFILE.FILE[2:2])

:              COMMENT  INSERT A LEADING ZERO IF NEEDED.
:              IF   N < 10  THEN
:                   SETVAR  N0 = '0!N'
:              ELSE
:                   SETVAR  N0 = N
:              ENDIF


:              COMMENT  IF NRJEHUB, THEN STREAM MSG FILE NOW.
:              IF   N = HPSYSNUM  THEN
:                   !TEL !HPDATEF !HPTIMEF  - JHPNNMON      LOCAL STREAM
:                   STREAM HP!N0.NRJEJOB,]
:              ELSE
:                   COMMENT  CHECK FOR SONJOB ALREADY RUNNING.
:                   IF   JSCOUNT ('JHP!"N0"MON,NRJEJOBQ.SYS') = 0  THEN
:                        !TEL !HPDATEF !HPTIMEF  - &
JHPNNMON      STREAM   JOB JHP!'N0'MON.
:                        STREAM JHP!'N0'MON.NRJEJOB.SYS
:                        !TEL !HPDATEF !HPTIMEF  - &
JHPNNMON      STREAMED JOB JHP!'N0'MON.
:                   ELSE
:                        !TEL !HPDATEF !HPTIMEF  - &
JHPNNMON      JOB EXISTS JHP!'N0'MON.
:                   ENDIF
:              ENDIF
:         ENDIF
        FORFILES  HP##.NRJEJOB

:       !TEL !HPDATEF !HPTIMEF  - JHPNNMON      WAIT !W SECONDS.
:       CALC PAUSE (!W)
:  ENDIF
 ENDWHILE

:COMMENT  END JHPNMAIN
```

```
:COMMENT  MPEX CMD FILENAME = JHPNNMON.AACONFIG
:SETVAR  WSID = 'MVS1R101'
:SETVAR  W = 90
:SETVAR  TEL = 'TELL LOGECHO.SYS'
```

```
!JOB JHP01MON,NRJEJOBQ.SYS,NRJE;OUTCLASS=,1;PRI=CS;INPRI=12
!
!COMMENT  I AM STREAMED BY JHPNMAIN AND RUN ON THE NRJE HUB NODE.
!COMMENT  I WATCH MY HPnn.NRJEJOB.SYS FILE AND FORWARD THAT DATA TO
!COMMENT  MY TARGET NODE.
!COMMENT  I ALSO WATCH THE JOB QUEUE LOOKING FOR MVS1Rnnn,NRJEMON.SYS
!COMMENT  TO LOG OFF.
!
!TELL LOGECHO.SYS JHP01MON - LOGON COMPLETED. START MPEX...
!
!CONTINUE
!RUN MPEX.PUB.VESOFT;INFO="XEQ JHPNLOOP.NRJEJOB"
!
!TELL LOGECHO.SYS JHP01MON - LOGGING OFF...
!EOJ
```

```
:COMMENT  FILE = JHPNLOOP.NRJEJOB.SYS
:COMMENT  I AM CALLED BY JHP##MON.NRJEJOB.SYS

:SETVAR  MPEXCMDTRACE = 0
:SETVAR  HPAUTOCONT = TRUE

:!TEL !HPDATEF !HPTIMEF  - !HPJOBNAME      MPEX ACTIVATED.

:SETVAR  N = INTEGERPARSE (HPJOBNAME[3:2])
:COMMENT  INSERT A LEADING ZERO IF NEEDED.
:IF   N < 10  THEN
:     SETVAR  N0 = '0!N'
:ELSE
:     SETVAR  N0 = N
:ENDIF


:COMMENT  WATCH THE JOB QUEUE FOR !WSID TO LOG OFF.  ALSO,
:COMMENT  FORWARD THE DATA TO THE TARGET NODE IF NEEDED.
:SETVAR  JHPNLOOP_INFINITE = 1
 WHILE   JHPNLOOP_INFINITE = 1
   XEQ JHPNNMON.AACONFIG

:  COMMENT  IF NRJRMON IS DOWN, THIS IS THE LAST TIME THRU.
:  IF   JSCOUNT ('!WSID,NRJEMON.SYS') = 0  THEN
:       SETVAR  JHPNLOOP_INFINITE = 0
:       !TEL !HPDATEF !HPTIMEF  - !HPJOBNAME      !WSID HAS LOGGED OFF.
:  ELSE
        REPEAT
:         IF   RFILE.EOF > 0  THEN
                 XEQ JHPNXMIT.NRJEJOB
:         ENDIF
        FORFILES HP!N0.NRJEJOB
```

```
:       !TEL !HPDATEF !HPTIMEF  - !HPJOBNAME      WAIT !W SECONDS.
:       CALC PAUSE (W)
:  ENDIF
 ENDWHILE

:COMMENT  END  JHPNLOOP
```

```
:COMMENT  FILENAME = JHPNXMIT.NRJEJOB.SYS

:COMMENT  I AM CALLED BY JHPNLOOP.
:COMMENT  I ATTEMPT TO LOG ON TO THE TARGET NODE.  I KEEP TRYING
:COMMENT  SINCE THEY MAY BE IN A FULL BACKUP.
:COMMENT  ONCE THE REMOTE LOGON IS OK, I FCOPY THE MSG FILE TO A STD
:COMMENT  DISC FILE.  I THEN DSCOPY THIS FILE TO THE TARGET NODE.
:COMMENT  IF THE DSCOPY FAILS (POSSIBLY DUE TO A PARTIAL BACKUP ON
:COMMENT  THE TARGET NODE), I INCREMENT THE TARGET FILENAME AND TRY
:COMMENT  AGAIN.
:COMMENT  I THEN STREAM THE TARGET FILE AND LOG OFF.

:COMMENT  OPEN THE DSLINE.  KEEP TRYING (RMT MAY BE IN FULL BACKUP).
:SETVAR JHPNXMIT_INFINITE = 1
:WHILE  JHPNXMIT_INFINITE = 1
:  !TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   ATTEMPT TO OPEN DSLINE.
:  SETJCW  CIERROR = OK
:  CONTINUE
:  REMOTE HELLO MIS!HPSYSNUM,NRJEJOBQ.SYS,NRJE;&
     HIPRI;DSLINE=NRJE!N=MIS!N
:  IF   CIERROR <> OK  THEN
:       CALC PAUSE (W)
:  ELSE
:       SETVAR  JHPNXMIT_INFINITE = 0
:  ENDIF
 ENDWHILE
:!TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   DSLINE NOW OPEN.


:COMMENT  MAKE SURE THE REMOTE TARGET FILENAME IS FREE.  KEEP TRYING
:COMMENT  (MAY BE IN PART BACKUP).
:SETVAR JHPNXMIT_INFINITE = 1
:WHILE  JHPNXMIT_INFINITE = 1
:  !TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   ATTEMPT TO PURGE TARGET.
:  CONTINUE
:  REMOTE BUILD HPX!N0.NRJEJOB
:  SETJCW  CIERROR = OK
:  CONTINUE
:  REMOTE PURGE HPX!N0.NRJEJOB
:  SETVAR  CIERROR = CIERROR

:  IF   CIERROR <> OK  THEN
:       CALC PAUSE (W)
:  ELSE
:       SETVAR  JHPNXMIT_INFINITE = 0
:  ENDIF
 ENDWHILE
:!TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   TARGET IS PURGED.


:COMMENT  RENAME GENERATION FILES.
:SETVAR  GEN = 9
:PURGE HPX!'N0'G!GEN.NRJEJOB
 WHILE GEN >= 2
   RENAME HPX!'N0'G![GEN - 1].NRJEJOB,HPX!'N0'G!GEN.NRJEJOB
:  SETVAR  GEN = GEN - 1
 ENDWHILE


:COMMENT  FCOPY FROM MSG FILE TO STD FILE.
:!TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   FCOPY MSG TO STD.
:PURGE     HPX!'N0'G1.NRJEJOB
:FILE  STD=HPX!'N0'G1.NRJEJOB;STD
:FCOPY FROM=HP!N0.NRJEJOB;TO=*STD;NEW
:ALTFILE HPX!'N0'G1.NRJEJOB;SQUEEZE


:COMMENT  DSCOPY STD FILE TO TARGET NODE AND STREAM.
:!TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   DSCOPY STD TO TARGET.
:DSCOPY HPX!'N0'G1.NRJEJOB;HPX!N0.NRJEJOB,NRJE!N;COMP
:!TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   REMOTE STREAM.
:REMOTE STREAM HPX!N0.NRJEJOB,]
:COMMENT  PURGE TO PREVENT THE FILE FROM BEING CAUGHT IN A BACKUP.
:REMOTE PURGE HPX!N0.NRJEJOB


:COMMENT  LOG OFF REMOTE.
:REMOTE BYE
:!TEL !HPDATEF !HPTIMEF  - JHPNXMIT.!N0   REMOTE BYE.

:COMMENT END JHPNXMIT
```

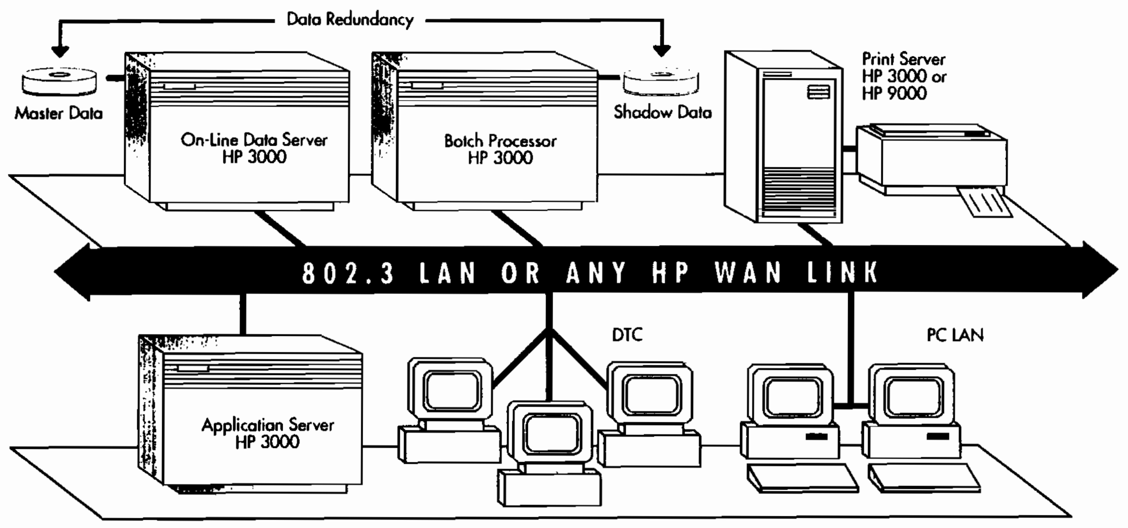
Networking & HP3000 Computer Systems

During this session we will be reviewing how Spectra-Physics Scanning Systems integrated four HP3000s into a single cooperative processing unit. We will cover how the need to increase our computing capacity (CPU) along with the need for a disaster recovery CPU lead us through the normal Hewlett Packard replacement review process. the typical solution, adding a larger HP3000 CPU, did not fit within our budget limits. Our solution was to keep our existing HP's and purchase smaller HP3000 computers and then network them together via Quest's Netbase networking software.

THE NEED FOR ADDITIONAL COMPUTING CAPACITY

In January of 1991 I was notified that the HP3000/950 was officially at the end of its useful life. HP had issued what should have been its death knoll! But, unknown to our HP sales reps, "things" were afoot that would extend the life of our 950 for at least two more years. Our major concern was the increasingly common problem of being out of CPU. While we had minor memory problems, the major problem was to heavy of a computing load and not enough CPU. We contacted our HP sales rep and began the process of determining what our options were. Like most other DP Managers, I requested a CPU that would have a useful life of about three years. As luck would have it, ASK/HP would be able to deliver one, an HP3000/960, within the time frame that we

THE NETBASE SOLUTION


Data Redundancy
Master Data
On-Line Data Server HP 3000
Batch Processor HP 3000
Shadow Data
Print Server HP 3000 or HP 9000
802.3 LAN OR ANY HP WAN LINK
Application Server HP 3000
DTC
PC LAN

NETBASE SERVICES :

Network File Access **Shadowing** **AutoRPM** **Spooling (NBSpool Plus)** **Statistics**

needed. The price for the "box" even fit within our budget. But, then came the "BIG" surprise! We are an ASK/ManMan shop, using PowerHouse and numerous "other" third party software packages. The cost to upgrade our major 3rd party software packages came in at about 1 1/2 times the cost of the hardware. That left us several hundreds of thousands of dollars short!

Oh, what to do?

I immediately requested some time with the V.P. of Finance to discuss the dilemma. After reviewing the problem several times with management, a decision was made. Spend no more than what was originally budgeted AND improve user response time before the end of the 3rd quarter. I was certainly glad I took the time to seek out his expert advice!

THE SOLUTION - NETBASE FOR HP3000 NETWORKING

In reviewing our options I consulted with my Operations Supervisor, Lois Andersen, and several of the Programmer/Analysts. No immediate solution could be found. How do you get more into an already full box? While several application solutions could reduce the system load, it did not relieve our major problem periods. Then, Lois remembered seeing a product that linked HP3000 CPU's together into a network of sort. After the information she had on Netbase (the software), we decided to contact Quest, the company that markets Netbase. With the placing of this call began what has proven to be a very interesting and rewarding project!

After Lois discussed our problem with Quest, they thought that Netbase could provide us with a solution that fit within our budget. Ron Vangel, the marketing rep from Quest, arranged to present an overview of what Netbase could do for us here in Eugene.

WHAT EXACTLY IS NETBASE?

Netbase is a software product that utilizes the Network Services (NS) featured on an HP3000 to communicate between HP3000 "nodes". While DS and NS services have been available for years on HPs, the ability to use multiple CPUs as a cooperative processing unit had not existed. I have worked in several HP shops that would have loved to have had the ability to "network" HP's! Normally when we spoke of cooperative processing between HP CPUs we were referring to passing files between systems by either tape or direct file transfer, and then running a program on the system that the file was transferred to. But, with Netbase we are much closer to true networking for HP's.

The "heart" of Netbase is its dictionary. The dictionary defines how the "network" will work. You specify where files, programs and users reside (one copy for each networked CPU). Once the dictionary is defined, the Netbase program is installed in either the system or account XL. Then, for every call for a program or file, Netbase will first check

Networking Building Blocks

The Link	-	The physical connection between two or more computers.
Net IPC	-	A programmatic interface used to communicate via various hardware network links.
NS	-	HP's product that is built using NetIPC to provide networking services including VT, RFA, RDBA and NFT.
NetBase	-	Like NS, NetBase is built using NetIPC. It provides an improved version of RFA/RDBA as well as additional functionality.

NETBASE

SAMPLE DICTIONARY

```
Netbase Directory Program [0.9.2) Update 3 (C) QUEST Software 1987

D> LISTF @.@.@,DSET      List Files
Directory: NBB5.DATA.NETBASE

----LOCAL REFERENCE-----    --------REMOTE EQUIVALENT---------   (FILE STATUS INFO)
FILE      GROUP     ACCOUNT FILE        GROUP     ACCOUNT        NODE

@         ABC       ASK -->       (SAME)                         RSD948

FINDB     FDATAB38  ASK --> FINDB       FUK       ASK            RSD



D> LISTL @.@.@      List Logon User List
Directory: NBD5.DATA.NETBASE
--- LOCAL LOGON ---      ---TO---       --------- REMOTE LOGON -----------------------
User      Acct           Node           Logon String

@         ASK            RSD948         ,ASKUSER/.ASK.USERQUIZ
ALLEN     ASK            RSD948         NB,ALLEN/.ASK,USERQUIZ
```

```
D> LISTP @.@.@       List Programs to be run on remote nodes
Directory: NBD4.DATA.NETBASE
------Local Program---------        ---------Remote Equivilent---------     Node
Program    Group      Account         Program    Group      Account

TR397X     MPUB       ASK             TR397      MPUB       ASK               RSD948
@          @          MANRSD60                   -- SAME --                   local



D> LISTS @.@.@,INFO      List file statistics
Directory NDB5.DATA.NETBASE

--------TRANSFORT REFERENCE-----------        -----LOCAL COPY REFERENCE--------  FILE INFO
FILE       GROUP          ACCOUNT              FILE       GROUP          ACCOUNT  -SYNC
                                                                                  - QUE
FINDB      FDATAB31       ASK                             ---SAME---              - USR
                                                                                  - DIS
                                                                                  - SHR
                                                                                  - LOG
```

if the file or program is included in the dictionary. If an entry is found, the call is transferred to the node (CPU) specified. The receiving node checks the incoming request against its local dictionary and then completes the process.

WHAT ARE THE COMPONENTS OF NETBASE?

Netbase is composed of five different features all defined in a central dictionary. These features are:

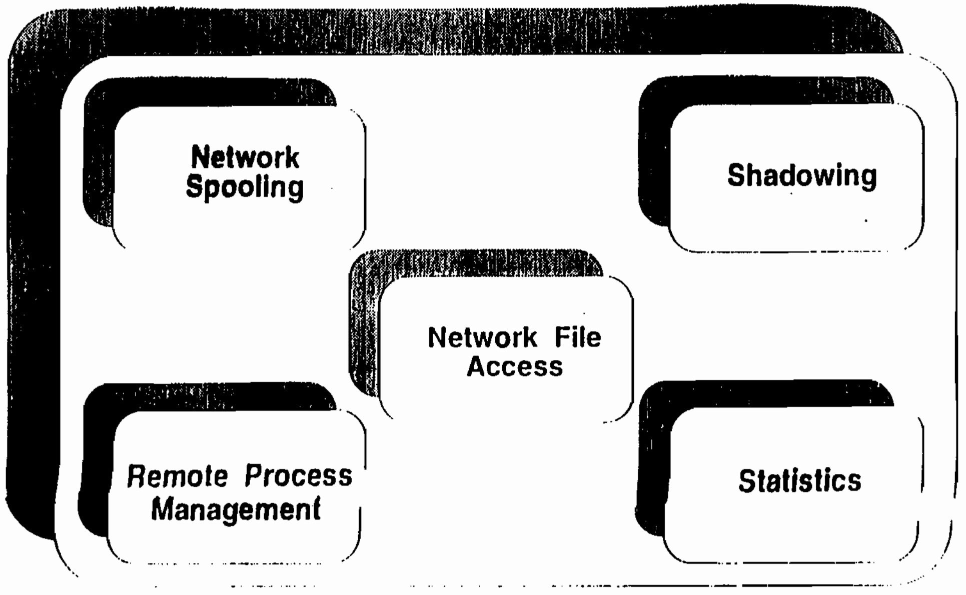
1) Distribute your files across multiple HP CPUs.
2) Run programs from any HP "node".
   (A node is the name you assign to each HP in your network.)
3) "Shadow" or mirror update transaction across the network.
4) Remote direct spooling.
5) Statistics reporting.

The key to all Netbase features is in its dictionary. In it you specify which node your files, programs, printers and users are on.

HOW DID WE USE NETBASE?

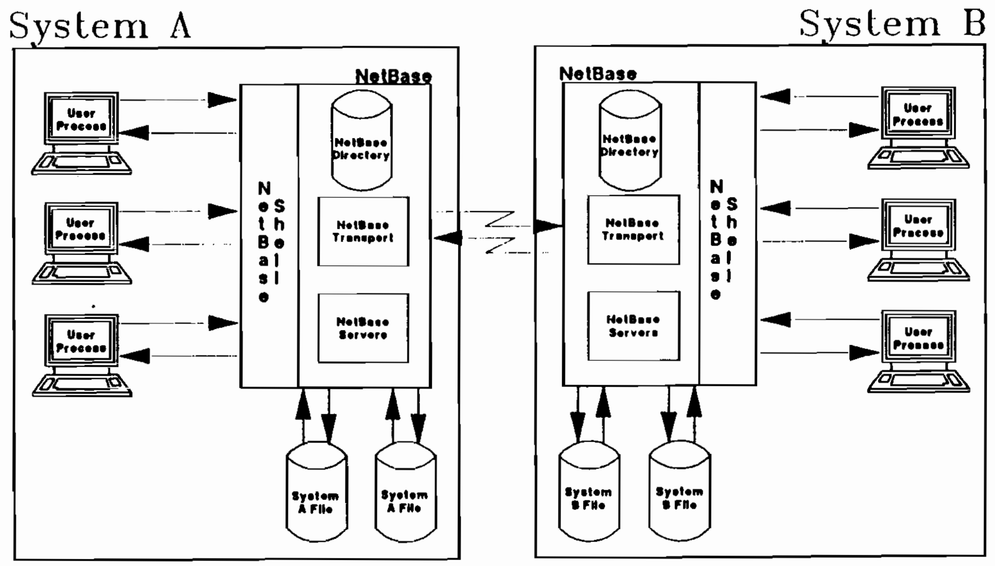
Our computing environment in January, 1991 was comprised of two HP's. One 950 and a 922. The 950 was used for our ASK software, PowerHouse and about ten other

NetBase Feature Overview

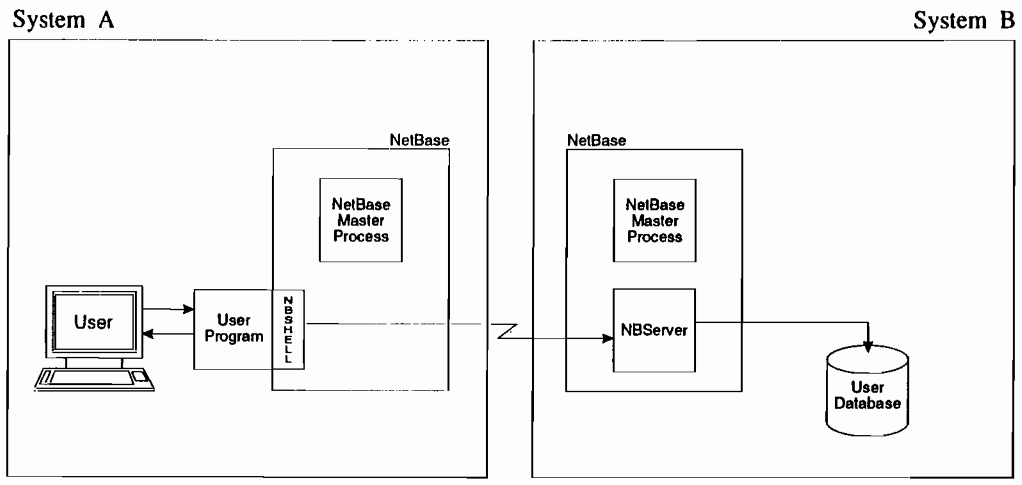


Network File Access

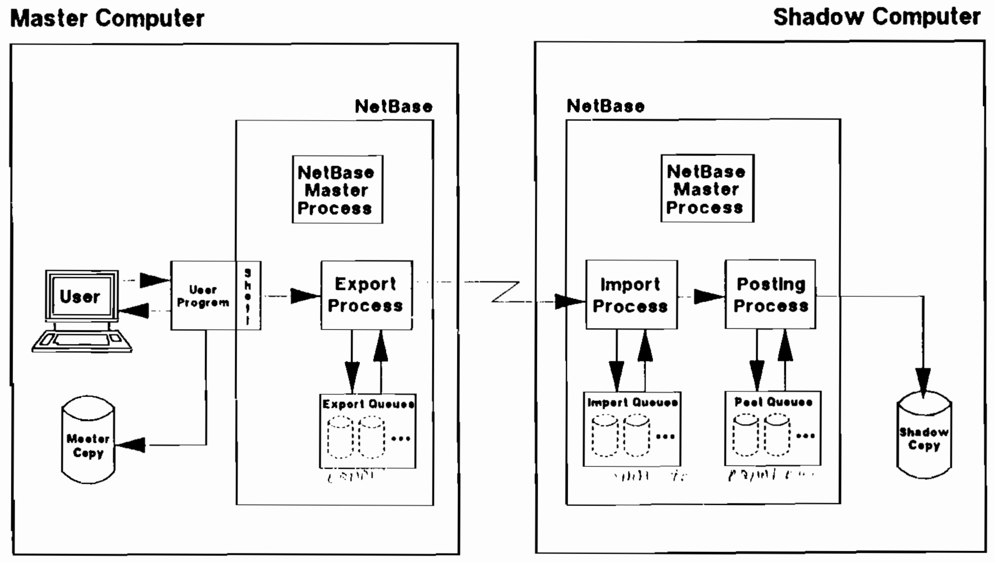
Overview



NetBase
Network File Access



Shadow Transport



software products. The 922 was used for a custom system written with PowerHouse that helps us build and track our configurable scanners. Neither machine had any additional computing capacity. So, what to do? Luckily, an HP3000/925 and HP3000/70 would soon be available from another division. We had the 925 shipped to us immediately. When the 925 was installed, we configured it to handle ALL spooling from the other CPUs and moved all our printers to it. Per Quest's measurements, spooling can take as much as 15% of a CPU. With remote spooling on, spooling services overhead on the 950 & 922 CPUs were virtually eliminated. We also moved one of our more CPU intensive applications to the 925 to help eliminate periodic bottlenecks. With these changes installed, the frequency of CPU overloads on the 950 was significantly reduced, primarily due to the off loading of spooling to the 925.

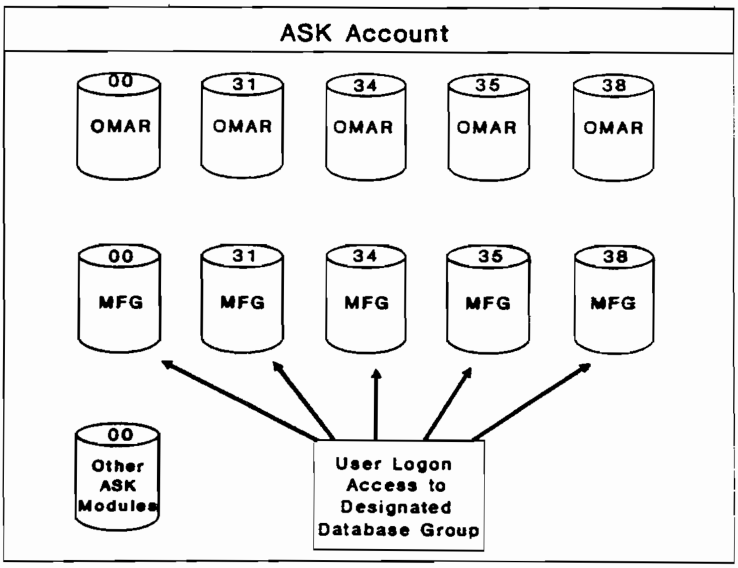
Next we took advantage of what then looked like a very generous trade-in program, trade the HP3000/70 for a new HP3000/948. We received the 948 in July and immediately began the process of "splitting" our ManMan software across the 950 & the new 948. We held several meetings with the Applications and Operations groups trying to decide how to best split our ASK applications for control, and the most efficient network utilization. As part of this project, Lois contacted other Netbase/ASK users to see what they had done. In the end, Lois decided that the best solution was to put OMAR and most of the financial applications on the 950 and Manufacturing and A/P on the 948. The Applications group, while not entirely happy with the split, agreed that they could live with it.

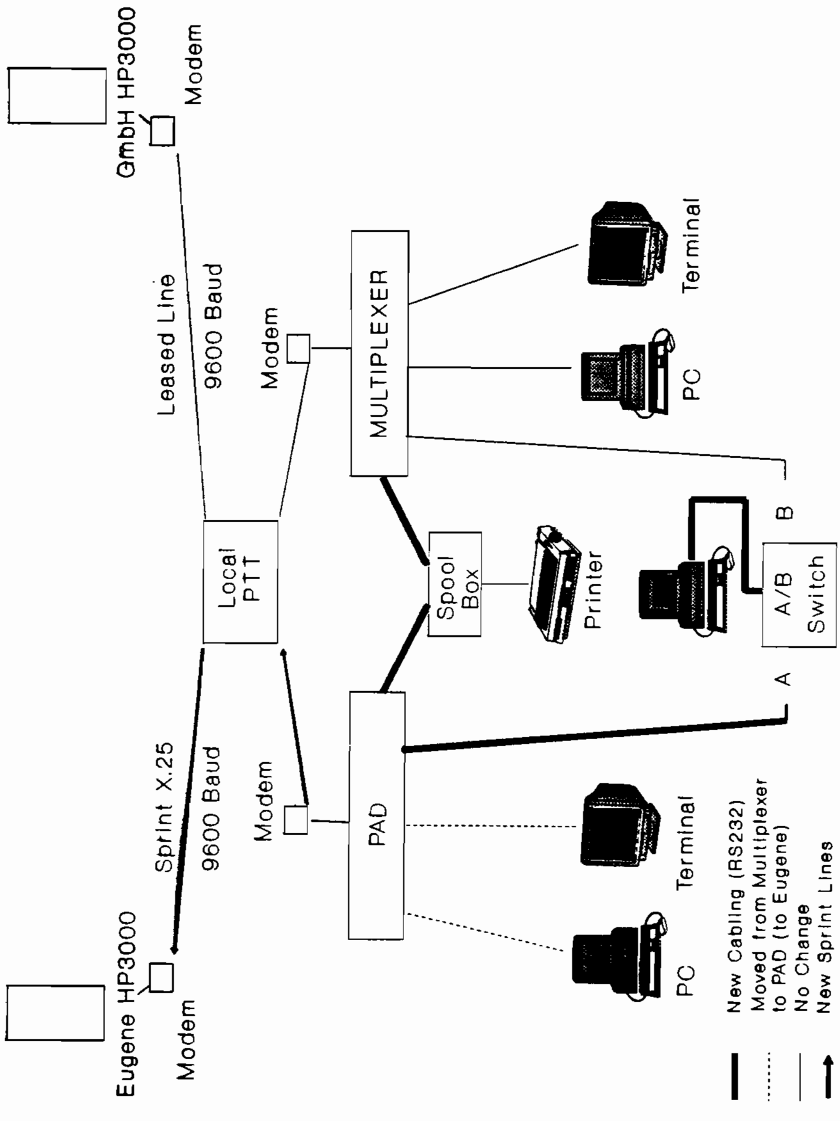
SPECTRA-PHYSICS SCANNING SYSTEMS

Account Strucuture

- Everyone uses the same account, ASK
- Data segregated by database group numbers

00	SPSS
30	Worldwide
31	SP GmbH
34	SP SARL
35	SP UK
38	SP SRL
40	SP KK] Future Additions
41	SP PTY] Future Additions





GmbH HP3000
Modem
Leased Line
9600 Baud
Modem
MULTIPLEXER
Terminal
PC
B
A/B Switch
A
Local PTT
Spool Box
Printer
Sprint X.25
9600 Baud
Modem
PAD
Terminal
PC
Eugene HP3000
Modem
New Cabling (RS232)
Moved from Multiplexer to PAD (to Eugene)
No Change
New Sprint Lines

The effort of the Operations group to implement the change-over was sizeable! In splitting the ASK and our other software across four CPUs they had to create intersystem maps for file locations, program locations, users and printers. Once they clearly understood where each of the elements would "live" they began developing the Netbase dictionary, and "stub" programs needed for the actual implementation. When this phase was completed, Larry Meston, the Systems Analyst on the project, modified our user menus to accommodate the "stub" program changes. This phase of the project took almost 2 weeks. But, once completed the dictionary worked very well. Yes, we did find some "gotchas", but overall the dictionary worked very well!

Another aspect of this project was how to create a disaster recovery capability. To accomplish this we created a "mini" computer room in our second building and added this location to our company wide network. We then moved the 922 to the new computer room. Once the 922 was installed in the remote building we began to "shadow" our production data file updates to it (from the 950 and the 948). We also shadowed file updates from the 922 to the 950. In the event that the computer room in either building is ever damaged, we can bring up a basic system using HP's in the other building (with planned upgrades for the 922/925 to be placed when a disaster occurs). For those who have seriously looked into a "hot" site for disaster recovery services, the cost can easily be $30,000 per year. With this option we are able to use our existing production HP's to provide us with the needed disaster recovery services.

As mentioned earlier, the Applications group was not thrilled about this project. I still am not clear as to what their real concerns were. In discussing the project, their main concern seemed to be Netbase and how it would affect their daily support efforts. Despite frequent assurances that it would work, (it WAS working elsewhere, even HP uses it!) they were not convinced! With some history now behind us Netbase DOES add some overhead, especially in problem isolation. While this extra step is needed to determine if Netbase is involved, it requires only minimal effort! I have now concluded that it was more of an emotional issue rather than technical issue.

The next battle was the additional complexity that Netbase created. While the execution of the NetBase processes is transparent to even the programmers, it DOES make the environment they operate in more difficult.

INSTALLING NETBASE

In order for Netbase to "trap" file requests, it needs to be installed at either the account or system level. Netbase will then trap file requests, route them through the dictionary, and to the correct "node". For this to happen a program MUST be run. This creates two problems. First, some programs, i.e. those that use PrivMode, may not function properly through a Netbase intercepted file call. These types of programs need to be SPECIFICALLY identified so that the NetBase system will exclude them. The next problem can be described as the "I can use it, but I can't see it" syndrome. MPE command interpreter commands are NOT programs. As such they are not "seen" by NetBase. For example, a program opening SOEFIL, will work just fine. But, if you do a LISTF against the file, MPE will return a "file not found" message. This is because the file "lives" on another node. The LISTF does not SEE the file because the LISTF command does not open a file. This same problem exists for most other MPE commands.

Another consideration that needs to be included in how you place files on the network is the size of the file. Even though NetBase is very efficient, data is still passed to the node where the program executes. As such, you need to be careful that serial reads of large files are NOT done by programs on remote nodes! If they are, expect to see significant increases in the run times for these programs. NetBase offers two alternatives to eliminate the data traffic on the network, but both require some thought and setup. One option is to create a stub program. A stub program is simply a label that is used by NetBase to determine on which node the "real" program is. If your large file (which is serially read) is on a remote node, use the stub program approach. The next option is to shadow the remote file on the local node. NetBase will know that the file exists there. For read only access to files, Netbase will first look at shadowed files on that node. Note that for keyed reads and for serial reads for small files, this should not be a concern.

The final area we needed to understand was how to control Netbase in our environment. The procedures you need to use are actually simple, the problem is remembering to follow them EXACTLY! Netbase can be started or shutdown by node. If one node is down, either because you shut it down or because of system/network problems, nodes that are shadowing to the downed node "buffer" their requests until the node has Netbase restarted. If the data files are needed for batch or on-line processing then Netbase will be not function until the network node services are restored.

OUR CURRENT CONFIGURATION

Today, SPSS has four HP3000 CPUs; a 950, 948, 925 and a 922. They are ALL working together to support our processing and disaster recovery needs. Originally, we had also considered using NetBase to track OMAR & inventory information from our worldwide sales locations (they were on separate remote HP's). We actually implemented the process to shadow OMAR & inventory file updates for these sales offices to our local CPU's, but later decided to process their orders directly on our local HP's. When the dust finally settled, we had increased our original computing capacity (CPU) by over 100% & stayed within our capital budget. Our pre NetBase computing index was, using the 950 as a 1, was a 1.6. After the new CPUs were added, our computing index rose to a 4.2. We now have a comfortable margin of computing capacity. This has allowed us to incorporate our foreign subsidiaries processing directly onto our Eugene HP's, and eliminated the CPU overload conditions that were common during most of 1990 and early 1991.

SUMMARY

We have been using Netbase's network services since March 1991, and are very pleased with the product and the service we receive from Quest. As in most products, there are always "things" that we feel could be done better. Quest has been very responsive to all the key issues we have had, and have been better than average for other non critical issues. The use of Netbase provides me with the ability to extend the life of my existing equipment, and may allow us additional savings in the future by not forcing us up to larger single systems solutions. Until the software world rethinks their marketing strategy for pricing their products, Netbase offers us, the end user, a viable, economic method of controlling how we increase our computing resources.

One question that I always ask end users is "Would you buy the product again"? In the case of Netbase, the answer is a resounding YES!

EPILOGUE

In 1993 we will be consolidating our four HP3000's back to two (an HP3000/948 and an HP3000/957). We will continue to use Netbase on the remaining systems. Netbase will continue to be used for the foreseeable future at SP. Without it, we would be forced into purchasing an HP3000 that would fall into a higher class for our software vendor's. This would result in both a higher hardware and software expenses. Netbase is again providing options that provide us with BETTER cost competative options!

Paper# - **3351** **Object Oriented Applications** by Dr. David Johnson

Johnson Computer Software Team Limited, Suite 5012, 3080 Yonge Street, Toronto, Ontario, Canada M4N 3N9

(416) 487-3631

Object Oriented Programming or OOP has been a major topic for several years now. Now Object Oriented Databases are making an appearance. Both OOP and OODBs are system development tools used pretty much exclusively by system designers and programmers. What about the end user dealing with screens and fields, records and input modes?

The importance of a good user interface is now pretty much universally recognized. Almost as universal is the acceptance of GUIs as the state of the art in user interfaces. But a good user interface alone doth not a system make. It doesn't necessarily even make the system easy to use.

There are, unquestionably, substantial advantages to developing systems with a standard GUI - reduced training time, the availability of multi-tasking and so on - or even just putting a (good) GUI in front of an existing system. But more can be accomplished by looking more closely at the user's needs.

The introduction of NewWave in Dec 1987 prompted us to look at Windows as the underlying user interface. For all of the reasons that one considers GUIs a good thing, we undertook a development project to put a Windows front end onto an existing mini based application. The result was a client/server application with an HP3000 as a database server and data processor with a PC running Windows providing the front end. It was certainly successful from the point of view that it provide the end users with a system that had a common user interface hence easier to learn, more functionality than the previous character based interface and improved performance by off-loading certain processing from the mini. However it was not a complete panacea: it was not completely intuitive as the user still had to be conscious of the fact they were dealing with and manipulating records from the host database. At approximately the same time we embarked on a project to implement NewWave and addressed the issue of migrating host based applications running on terminals to the NewWave environment.

What became very clear from our Windows project was that we were missing something in the quest to give the user a seamless view of the system. Users working in the NewWave environment who had to also use the host based applications had to adjust their thinking as they switched from NewWave objects to working with host based systems dealing with records.

One of the fundamental goals of the system designer is to create a system that users can get the most *use* out of it. It doesn't matter how much functionality is built into a system if the users can't find it or exploit it. When users learn a computer system, they build a mental "model" of how the system works. They must then map their real world objects/documents/working materials to this model and back again. The ultimate success of the system will depend on a couple of factors:

- How easily the model is "constructed" in the user's mind is a measure of easy it will be to train users.
- How "faithful" the model is, that is how well the system behaves or conforms to the user's expectations, will determine how easy the system is to operate and how easily the user will be able to use it to cope with new or different situations and be able to get information from the system.

In short the success of the system is measured in terms of how easily the users can make it do what they want it to do.

Implementing NewWave highlighted one very important fact: *anybody* can be taught to use NewWave. The biggest obstacles for first time users was developing the hand/eye coordination to use the mouse. Once past this, the environment is so intuitive that even Luddites can be taught to use a computer. NewWave is the only environment that I have seen where this is true.

A classic illustration of this was introducing a user to HP's NewWave Mail. To mail something, the user was told, drag the icon to the Out Tray. The system builds an outgoing envelope complete with distribution list, puts the object in it and opens the distribution list for the user to fill in the intended recipients. So far nothing could be easier. The user then asked what could be done to facilitate mailing things to a predetermined list of people. Enter the distribution list object - we had provided a master distribution list with the names all the people in the company. It was immediately obvious to the user, without being prompted, that he could make a copy of this distribution list (basic NewWave drag and drop technique) and that he must be able to open it, edit out some of the names, close it and give it a different name to create any customized list for his own needs. The reason it is so easy for a user to make use of this application is that the presentation of the system to the user so closely resembles the real world "paper" mail. People understand In Trays and Out Trays, envelopes, distribution lists and mail trucks. By presenting the system to the user as these types of objects, the user can very easily and accurately build a mental model of how the system works.

Our challenge, to be able to successfully migrate host based applications, was to find a way to present the user with a system that closely resembled the real world documents[1] they work with. Moreover the system should operate on these documents in a very canonical fashion. Visions of Object Oriented database management systems manipulated by Object Oriented programming languages then came to mind. However a fundamental time and cost constraint was to preserve anything and everything worth preserving of the existing systems. It was unrealistic to consider redesigning the applications from the ground up on the host and equally unrealistic to consider moving larger applications to the PC as NewWave applications (even the largest PC is not going to rummage through a few

[1] The term "documents" is not limited to pieces of paper but includes any job related items - spreadsheets, voice mail messages, video images - the user works with.

thousand NewWave objects with any alacrity to produce a report- something an HP3000 makes short work of).

What we needed was a way to take data in an IMAGE database and somehow present it to users as familiar objects - which they could manipulate in the very intuitive fashion NewWave objects are manipulated.

The remainder of this paper is the description of a couple of examples of applications first developed, years ago, on an HP3000 with a terminal interface. One is a relatively simple Art Inventory Control system and the other a complex advertising agency application. Both are multiuser, i.e., they must allow for simultaneous access by multiple users, both have compute intensive functions that involve manipulating large amounts of data in the database, and both have batch processing needs that also requires manipulating large amounts of data. In short they are not candidates for running on a PC.

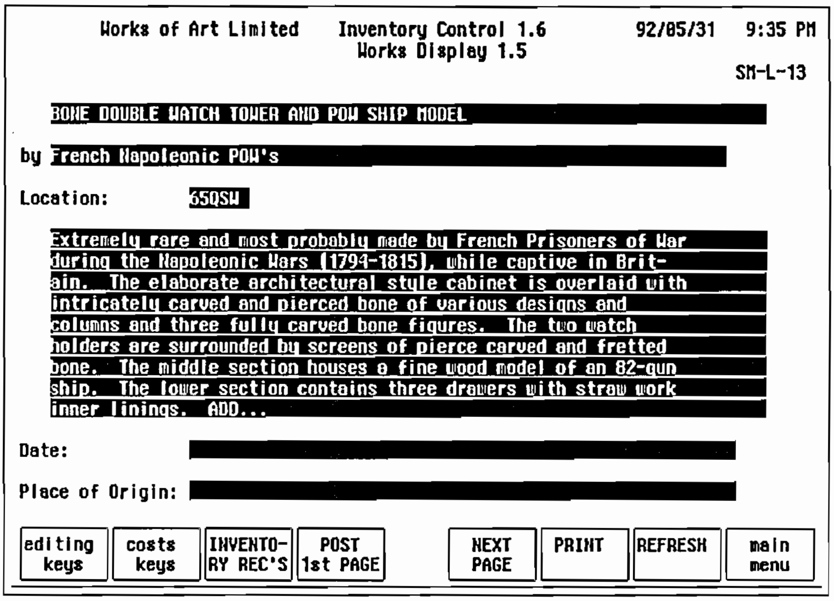
Art Inventory Control System

The first application, the Art Inventory Control system, developed a number of years ago on an HP3000 using an IMAGE database and terminals, actually had some "cooperative computing" in it's execution. It had the standard requirements of on-line entry, enquiry and modification of information about the individual works of art in the collection. Various accounting functions such as allocating insurance premiums to all the works at particular locations were built and a variety of batch reports developed.

One area of reporting, however, could not be satisfactorily met with by writing programs on the mini-computer. A master book containing detail pages of each work was maintained for the entire collection. This master list is, in the minds of those who are involved with the collection, synonymous with the collection itself. While there is a particular painting hanging on the wall of one of the galleries there is information about the history, origins and even some financial information on the corresponding detailed sheet that is considered as an important part of the collection. Moreover the list is not static: works are bought, sold, moved, restored etc., all of which requires updating the appropriate detail sheet from information kept current in the database. Because of the prominence of these detailed sheets their formatting is of primary importance. In the early '80s it was not feasible to create reports off an HP3000 that did things like justification of text on the page for a proportional font. The "solution" was to create a "report" to disc that was a file in HPWord format. The user would then use HPWord to call up the file and print it on a laser printer in a reasonable approximation of what was desired.

Migrating to a new HP3000 meant abandoning HPWord (which could not handle the newer fonts available for the Laserjet printers anyway). For word processing we had moved to AmiPro under NewWave and used a similar technique of producing a "report" to disc in an AmiPro format. It was not possible with any reasonable amount of work to create *exactly* what the users wanted. From their perspective the detail sheets were the electronic embodiment of the collection so perfection was a serious consideration. One user took a particular work as an example and created an AmiPro document with the required information in *exactly* the format she wanted to see it so she could say "here, this is how I want it to look". The obvious question from this user was "why can't we just turn the whole inventory control system into a collection of AmiPro documents?". It would completely satisfy the one group of users who work with the detail sheets but would be totally unsatisfactory for others who dealt primarily with the accounting aspects. The thought of having to develop some sort of system for the people in the accounting department so they could allocate insurance premiums was more than a little daunting. Think of having a PC rummaging through thousands of AmiPro documents looking for and extracting financial information, doing the necessary calculations and then updating the appropriate documents! The same concern applied to creating reports from the system. Moreover, all of this functionality for the other users was already in place working effectively and efficiently on the HP3000. If it ain't broke we can't afford to fix it.[2]

To these users the system looked like:



```
        Works of Art Limited     Inventory Control 1.6         92/05/31    9:35 PM
                                   Works Display 1.5
                                                                           SM-L-13

    BONE DOUBLE WATCH TOWER AND POW SHIP MODEL

 by French Napoleonic POW's

 Location:          650SW

    Extremely rare and most probably made by French Prisoners of War
    during the Napoleonic Wars (1794-1815), while captive in Brit-
    ain.  The elaborate architectural style cabinet is overlaid with
    intricately carved and pierced bone of various designs and
    columns and three fully carved bone figures.  The two watch
    holders are surrounded by screens of pierce carved and fretted
    bone.  The middle section houses a fine wood model of an 82-gun
    ship.  The lower section contains three drawers with straw work
    inner linings.  ADD...

 Date:

 Place of Origin:

 editing   costs   INVENTO-   POST          NEXT   PRINT  REFRESH   main
  keys     keys    RY REC'S  1st PAGE       PAGE                    menu
```

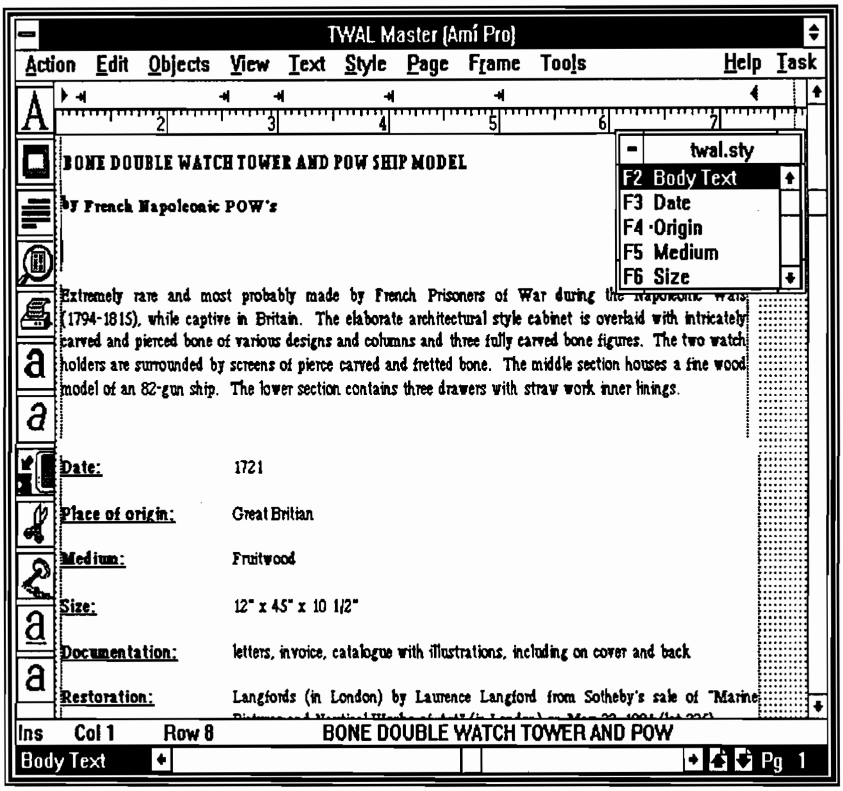
figure 1

Figure 1. shows a standard HP terminal running the character based application. The program turns on the "A" strap in the terminal to allow it to intercept all the cursor positioning keys so that changes are effected by positioning the cursor and typing the desired information. Programmable soft keys allow the user to perform functions and navigate through the system.

---

[2] Corollary 117 of Murphy's Law.

The way they wanted to think of the system was:



BONE DOUBLE WATCH TOWER AND POW SHIP MODEL

by French Napoleonic POW's

Extremely rare and most probably made by French Prisoners of War during the Napoleonic wars (1794-1815), while captive in Britain. The elaborate architectural style cabinet is overlaid with intricately carved and pierced bone of various designs and columns and three fully carved bone figures. The two watch holders are surrounded by screens of pierce carved and fretted bone. The middle section houses a fine wood model of an 82-gun ship. The lower section contains three drawers with straw work inner linings.

Date:	1721
Place of origin:	Great Britian
Medium:	Fruitwood
Size:	12" x 45" x 10 1/2"
Documentation:	letters, invoice, catalogue with illustrations, including on cover and back
Restoration:	Langfords (in London) by Laurence Langford from Sotheby's sale of "Marine

Ins Col 1 Row 8 BONE DOUBLE WATCH TOWER AND POW

Body Text Pg 1

figure 2.

Figure 2. shows an Ami Pro document open in the NewWave desktop

The technical issue then was could we leave the existing system in place but present the information to the users as a collection of AmiPro objects? The users who work with the detail sheets could then manipulate them in a way that was as natural as possible to them. If there is more information to add to the historical description then they add it. If they want to print a sheet they drag it to the printer. If they want to change the font from Times Roman 11 to GCTimes 11 then they can change the font. If a particular piece has special formatting requirements that doesn't follow the norm or extra information to include then they make the necessary changes. If they want blue letters on the screen they can have blue letters on the screen!

With an HP3000 server and PCs running NewWave the answer was that we could. The fundamental feature of NewWave that makes it possible is "sharing" or "hot links". What this means is that you can have more than one view of a NewWave object and a view can be inside another NewWave object.

The users' view above is an AmiPro object with shared views of another object (called an "Art Object") inside it. In other words, the illustration above is from the implemented system! Viewing it in NewWave there is no apparent difference between a simple AmiPro document and one with links into an IMAGE database. Working with it is not really any more complicated than working with any other AmiPro document: the user can change the fonts, print it etc. like any other document. The difference is that some information does not need to be stored in the application database on the host while other information such as the title, artist and "index number" visible in the example is part of the "corporate" information in the host database. If the user wants to change the artist, for example, the system only allows artists names from a list of valid artists in the host database. The user clicks on the artists name in the document and NewWave opens up the Art Object. This has a drop down menu of valid artists names loaded from the host database. If the user selects another name then the Art Object passes a message back to it's host server to update the database accordingly.

User approache the system by opening an icon called an "Art Collection", with any 32 character description they want, and choosing Locate from the Action menu. They can then use a variety of parameters such as key words in the title to locate any particular work or groups of works in the collection.

Another interesting feature of the approach we have taken is that the system may contain other NewWave objects not directly part of the host application. Correspondence, video images or anything else relating to the collection can be stored in the system either with the individual work or as part of the collection of objects.

An Advertising System

The second example is a vastly more complex application: broadcast media buying in an advertising agency. Again the application was developed some years ago using an IMAGE database to store the data and terminals to access to access the information. With a complex set of reporting requirements and links to other systems (billing data is passed to the financial system) the feasibility of redesigning using an Object Oriented Data Base or moving it to a PC was out of the question. Looking at the system from the user's point of view the key issue was the type of document(s) the user deals with.

Media buyers deal with a number of documents but one fundamental one is a media contract. When buyers complete their negotiations with the station representative, a contract is issued to the agency. Look in a typical buyer's office and you will find a filing cabinet filled with filing folders full of contracts. The information in the contract must be entered into the system: the user must "map" the data in the contract format to the input screen format. When changes need to be made, which is surprisingly often, this mental mapping must take place in reverse: the user must locate the information displayed in a record format and make the appropriate adjustments. The challenge then can be stated very easily: take the "buy" data in the system and represent it as a contract "object". Like the Art Inventory system the user sees one or more media system icons on their desktop. He or she opens it just as with any other NewWave container object and locate the objects inside it. New contract objects are created just like any other NewWave object by going to the Create a New... menu or by making a copy of an existing one. They add a new contract to the system by dragging it to the appropriate media system icon. When he or she locates a contract in the system the user can manipulate it like any other NewWave object and here is the fundamental difference between the screen based system and an object oriented system.

In the screen based system it is certainly possible to present the user with a screen that looks like a contract (ignoring the fact that it won't all fit on a 24 line 80 column screen and some mechanism has to be devised to "scroll") this is not accomplishing the same thing. What if the contract was put into the wrong estimate? This could mean it must be removed from one database and entered into another. In the screen based system this would require complicated programming and then training the user what procedure to follow. The only other recourse is to delete the entries from one system and re-enter them in the other. In the object oriented system the user clicks on the offending contract, drags it out of the system it is in and drops it into the one it belongs in. This is *exactly* analogous to what users do with actual contracts in their file drawers: they take them out of the wrong folder and put them in the correct one (it's often harder to find a piece of paper in the file drawer than a contract object in the computer). Similarly if a contract is cancelled they just drag it out and drop it in the waste basket.

If there are similar contracts (e.g., a winter buy and a spring buy that are nearly identical), it is much easier, in the object system, to make up the first contract then make a copy (CTRL, click and drag) to create the second. Make the appropriate modifications to the copy (give it a new name too) and drop the contracts on the appropriate media system icons.

For at least one piece of functionality in the system, this approach actually simplified the overall design. Requesting reports and jobs that pass information to the financial system is a major part of the functionality of the system. In the screen based system the user must sit at a screen (or fill in a request form to be given to a data entry person who sits in front of the screen) and fill in the request parameters to have a job run. Because of the number of parameters this can be very tedious and error prone so we designed a mechanism that allowed users to build report "profiles". In simple terms they could make a request and store the request parameters in the database to be used again for subsequent requests. Of course this subsystem required the facility to copy old ones to make new ones, delete old ones etc.. No matter how hard we worked on the screen interface a certain amount of training was required to get users fully conversant with the system. Users also requested the facility to schedule requests thus further complicating the system.

In the object oriented approach we created a Report Request object - a standalone NewWave object that stores request parameters. The media system application recognizes these objects (they are installed with a particular NewWave property to allow this). It communicates with the host application to validate parameters - valid entries are presented in list boxes for the user to choose from and to log the requests. The entire mechanism of creating, storing, modifying, copying and scheduling report profiles on the host is no longer required. Users create new report requests just like they create any other object, with the Create a New ... option or by copying an existing one. They can call these objects whatever they want (up to 32 characters). They can organize them in any way they choose on their PCs creating File Folders to put them in. They can pass them around to other users via the mail. They get rid of old requests by dragging to the waste basket. Most important of all because these are NewWave objects they can be manipulated by the agent. Requests for jobs to pass information from the media to the financial system can be scheduled as far in advance as desired and then will take place whether or not the user is actually there. That particular activity can be scheduled to take place during off hours when there are no on-line users. The beauty of the system is that while there is less programming effort there is also vastly more functionality and the functionality is available to the users in an intuituively obvious way

Reusable Objects

The two applications described above have nothing in common from an application point of view but they (along with other applications being turned into object oriented applications) behave in this new paradigm in a remarkably similar fashion. In the design of these systems we wanted to generalize as much as possible (for all the generally accepted reasons). At first this looked like generating reusable code, i.e., having written code for an "Art Collection" an "Art Object" to go into it, the source code could be copied and modified to provide the basis for the "Broadcast Media System" with a "Contract Object" to go into it. But working with objects and the notion of containment and sharing led us one step further to the concept of building the systems using individual objects as building blocks to assemble the system. The only custom coding is for specialized objects unique to the application and these are limited to providing only that functionality particular to the application. Moreover, other Windows/NewWave applications "integrate" smoothly into this environment as appropriate. The best example is using Ami Pro as the primary user interface for the Art Collection. Of course, the user may choose any other applications, e.g., NewWave Write, that supports shared views. And with Windows OLE applications beginning to make an appearance more "interfaces" will become available.

Our base for our object oriented applications is an application, called an Archive, developed to store and share NewWave objects from PCs onto the server. This provides the PC to server communications and the basic functionality of storing and retrieving NewWave and (encapsulated) Windows objects. We refer to this as the "System Base Object" or SBO.

We created a "General Application Object", or GAPP, and generalized the PC Archive object to be able to recognize "it's" GAPPs and communicate with them. GAPPs are container objects (like File Folders) but they also contain a special application object - the object unique to the particular application. These we call special application objects, or SAOs.

Because this is a client/server application we had already developed a messaging protocol between PC and server program. This was generalized to pass messages from the SAO through to a son process that services the host application database.

The application on the host is conceptually very simple: it receives input from a message file and must either find data in the database or update the database. It writes the results of it's activity to a message file, that is, if it was asked to find data it returns the requested data (or an error message) to it's output datafile; if it was asked to update the database it returns a status indicator (or an error message). Typically these applications are relatively easy to write and test.

The SAO on the PC can be a simple Windows application that puts up a dialog box with data received from it's host server although we find the real advantage comes with endowing the SAO with the capability to share it's data with other NewWave applications (in future think Windows OLE) which then gives users more choice in how they will deal with the data. In one application where we allow shared views in our SAO the user may use Lotus 1-2-3 to manipulate numbers and share the "bottom line" numbers into the SAO. From there they are automatically put into the host database.

New systems are created by re-using the SBO object, albeit with a different icon and installed with different properties, and re-using the GAPP, which can be done in minutes. Custom development is restricted to the host server (which is cognizant of the application database) and the SAO which is tailored to the particular application. Because the GAPP is a container object, the user may put anything else at all in there along with the SAO. The Art Container GAPP may contain other word processing documents such as letters to dealers or may contain a colour image of the actual piece of art. The media contract GAPP could contain a scanned image of the actual contract that arrived from the station. It could also, assuming your LAN had the band width to support it, contain live motion videos of the commercials being run. In short the *users* can use the system in a fashion that closely parallels how they deal with the actual day to day objects - contracts, videos, correspondence - that occupy their work space.

Paper #3352

CLIENT/SERVER COMPUTING WITH MPE

Presented by:
Joseph C. Geiser
Insurance Data Processing, Inc.
One Washington Square
Wyncote, PA 19095-0137
(800) 523-6745 / (800) 345-1360 (in PA)

INTRODUCTION:

Much has been said over the past twelve to eighteen months on the subject of Client/Server computing. Typically, the term is used to denote the method of computing which employs multiple hardware platforms, one of which is usually a Personal Computer.

All of the major vendors have their positions and respective strategies and solutions on the subject. HP users have been fortunate in the fact that HP has made significant inroads and products available to address this issue. These products have been out, not for several months or a year, but several years now. HP has also built their strategy which should make it a leader in the 1990's and beyond.

The HP strategy, the NewWave Computing Strategy, utilizes not only PCs, but also Mainframes, Minicomputers and Business Servers, which are tied together with network schemes. The "System" of the 1990's will not be a "single box" but a "network". Data will reside on a machine or on a collection of machines. The application will utilize data from these machines and will be transparent to the end-user.

Client/Server computing from an MPE standpoint is not much different than this definition statement. In a typical HP3000 evnironment, corporate data resides on HP3000s, there may or may not be a mainframe such as a 43xx, 30xx or ES-9000. If a mainframe exists, there is either an SNA link between the HP3000 and the mainframe or quite a bit of "sneakernet" (data transfer via tape) going on. There are probably PCs running spreadsheets or small database applications - more than likely with duplicate data entry or with file transfers using AdvanceLink or Reflections.

To really make this a Client/Server shop, add the following statements to the above scenario:

- Sharing data from one or multiple hosts, accessed from a single workstation.
- Sharing resources between one or more hosts and workstations.
- A Common User Interface which is easy to use and easy to access.
- Extend the life of existing data, data structures and applications.

What this yields is:

- Workstations (PCs) which can access data on multiple hosts.
- Workstations running applications using data from multiple hosts in a real-time environment for read and write.
- All software has a common user interface which is consistent across all applications.
- Existing databases and data structures as well as applications can be used.

This is client/server computing from an MPE standpoint. Sharing the processing load and data storage/retrival between hardware platforms, utilizing those "under-utilized" PC MIPS, using corporate data resources at the desktop and integrating systems so that they work together. This is true client/server computing.

How can this benefit you? How can you take advantage of this environment? This does sound a bit futuristic and quite complicated to achieve - especially in an environment where applications are written and operate solely within the confines of an HP3000.

Yes - it does sound futuristic, and it does sound like you need to hire a staff of PC gurus to make it happen - but you really don't. Take a good look at your applications - or better yet, we can look at ours. When we're through, you'll see that you can use some or all of the same steps we used to implement Client/Server computing at your installation also.

Vision-MIS (A.02.xx) - The HP3000-based solution:

Our application, Vision-MIS, is a typical HP3000 application. The majority of the code was written in COBOL (with some TRANSACT). Reports are written with Business Report Writer (HP/BRW). It uses VPLUS for screens and TurboIMAGE as the DBMS. There is some KSAM and MPE file use. It has some batch processes but is predominately online and real-time.

There are more than three-million lines of code comprising the system and is maintained with HP3000 based tools (ie: OCS/Librarian, Documentation/3000), but it's still tough to maintain.

Since we serve the insurance industry - our clients are small to medium sized property and casualty companies. They either have small data processing departments with HP3000s or, like the majority of our clients, do not have any data processing capability (except PCs and local area networks) and rely on us for this function.

The insurance industry is highly regulated by individual states. Requests for data by internal users and governmental regulatory bodies are ever-changing. Rates need to be set by company actuaries. Data on losses (claims) is critical in setting these rates. State Insurance Departments look at both premium and loss data to assess the viability and solvency of a company.

We receive at least eight to ten requests each week for report modification or new reports to satisfy these needs. We receive several requests each month for state-specific modifications in data capture and retention to meet the regulatory requirements. Many of these changes come with minimal notice. We employ one full-time person just to produce reports with HP/BRW and we attempt to comply with regulatory changes in timely fashion. We meet these requirements most of the time, however, it is a herculean effort in most cases.

Then there are the "nice to haves..." The requests where the end-user says, "Ya know, it would be really nice if the system did ..." (you fill in the blank - it happens in EVERY application). We also support an extension for homewoners policies called policy rating, where information pertaining to a property is entered and the system computes the annual premium based on these factors. These systems are extremely complex and again, based on rules that change either by state governments, affiliated bureaus, or the companies themselves based on their own experience. These rating systems take at least nine months to a year to develop using traditional methods and most are custom to the client (meaning there are multiple systems being supported).

We immediately saw two areas we needed to address. One was *Reporting* and the other was *Rating Systems*.

We knew that our clients were using PCs with spreadsheets (Lotus is ever-popular) and databases (X-Base products such as *dBase, Foxpro and Clipper* - one was using *R:Base*). We thought that if the CLIENT could access his or her own data directly from our databases, selected it and downloaded it to their PCs for use with these packages (for reports and analysis), this would free up resources (both programming and machine utilization) for other areas in our systems.

We also knew thet our clients were beginning to use rating systems based on PCs, rating policies, then re-entering them into our systems. If we could somehow "link" these rating systems to our system, we could save the duplicate entry and not need to track all of the rating changes, as the third party and the client would be responsible for these changes. In addition - the third party company would focus on the rating aspects and could make these changes faster than we could ever do them. Most important, however, we could offload the processing of these policies from the HP3000 to the PC, thus reducing system load and improving performance.

PHASE 1 - Reporting:

Since our client's need for data is ever-changing and usually on short notice, we found ourselves with quite a "respectible" backlog - meaning that even though we could put through changes easily (with BRW), more change requests were coming in every day.

In addition, they were also asking for "file downloads" of data for their own use. This meant writing programs to create flat ASCII files which would then be downloaded with AdvanceLink or Reflections by the client or sent on diskette.

We decided that the client should have a means to access their data, using their selection criteria, and be able to use this data in the environment with which they are familiar. This meant that if the client uses Lotus for acturarial work, then the client should be able to access this data and use it in Lotus. The same held true with dBase (and similar) products.

NewWave Access was the solution to this problem. To test this scenario, we configured the databases of one client into the Information Access Server (on the HP3000), and used NewWave Access to satisfy one of his requirements. This task, to produce a letter to each insured with an inforce policy written by particular agencies. Setup of databases need only be done once on the HP3000. Once set, this acts as a "data dictionary" to the NewWave Access user.

In this requirement, we accessed the data using NewWave Access and exported the data to an R:Base database. The time required to perform this, excluding the setup time for the databases on the HP3000, which is done once, was less than an hour. We sent the diskette to the client, who set the letter up on his PC using the layout we provided. When the diskette arrived, he simply loaded the database to his hard drive, and ran the R:Base program. The letters were produced. This client now uses NewWave Access for most of his reporting requirements. Data is selected, sorted, and converted out to either Lotus or R:Base, where they then use the facilities in the application for reports or other tasks.

We now (as of this writing) have three clients using NewWave Access (as well as NewWave in general) with quite a few more to be added in the near future. We also expect to see our backlog start to disappear quite quickly.

One additional note to mention: Since NewWave Access can also access data on PCs, in addition to HP3000s, data which was already brought down or already resident on PCs, can be merged and used with other HP3000 data requests, thereby eliminating recurring requests for data.

PHASE 2 - The Use of NewWave

HP NewWave has been around for some time, and with 4.0 released, promises to be a real productivity booster. With the introduction of NewWave Access to our clients, we are also introducing NewWave and its capabilities.

NewWave Write is used with Lotus spreadsheets for reports within organizations. These are especially useful for one-page summary reports for company management. The use of OMF (Object Management Facility), spreadsheets created with NewWave Access can be "shared" or copied into NewWave Write documents. In addition, the Agent within NewWave automates these tasks so that they can occur on a scheduled basis (ie: every Monday or at the end of each month). We are anticipating the introduction of NewWave Mail to our clients in the near future so that these reports can be electronically mailed to their respective recipients, either manually or automatically by the Agent.

We have also brought in AdvanceLink for NewWave/Windows and have created a version of our online systems using "TermTalk" scripts. Now, instead of having to log on and have access to the MPE prompt, the user executes a task, which prompts for logon information (including passwords), performs the logon, and opens a folder with their appropriate functions. The user then drags a "script icon" (which may be labelled "Policy Inquiry") to the "terminal icon" (a minimized AdvanceLink) and drops it on the icon. The next screen they see is the HP3000 VPlus screen they have always seen. This is an interim step to a new version of Vision-MIS, which will be a full NewWave application and is currently "under construction" (more on this later).

Overall, NewWave has become the standard platform for our new clients. Existing clients are being migrated to the NewWave platform and should be completed by year-end, 1992.

PHASE 3 - Policy Rating

The rating of policies demands many machine resources and can keep an HP3000 very busy, to the point that performance will suffer. IDP has written rating programs for "personal lines" such as Homeowners (personal lines are those types of insurance which a "person" would purchase, as opposed to "commercial" lines which a business would purchase). IDP rating systems ran exclusively on the HP3000. Our problems here are that these systems ARE resource intensive and do impact on performance. Response time is unacceptably slow at times of peak demand.

Personal Lines rating proved to be a good subsystem as it takes quite a bit of patience to perform all of the calculations necessary to produce an annual premium amount on a policy. End-users for this subsystem usually do not relish having to code each factor and literally "looking up" premium factors and rates in a manual. Factors which go into rating a homeowners policy, for example, are the type of structure (brick, masonry, wood frame, etc), age of the home, location (inner city, suburbs, rural, etc.), and if the client writes in more than one state, which one. These are just a few - there are many more. These factors are loaded to tables on the system and the user enters values for each factor - the system would compute the premium.

Personal lines, however, is relatively easy compared to commercial lines. A business can not only insure property, but cars, crime coverage, glass, signs, liability, and so on. Glass, for instance, is rated on size and number of panes, as well as grade of glass. Combine these - and it can turn into a real nightmare to do by hand.

We felt that to try to "be all things to all companies" was impossible, and writing a commercial rating system would not only be very time-consuming, but would also not perform well on an HP3000. In addition, these systems were readily available at the PC level and our clients were using these packages to rate policies. The problem was that these policies had to be reentered to our system. Not efficient by any standard.

We made the decision to bring PC rating into our system. We wanted an integrated system which would utilize data we currently stored and would store policy data on our TurboIMAGE databases. We saw this as an opportunity to split the work between hardware, thus improving throughput. The PC-based system had to be complete and modular, able to process personal and commercial lines of insurance, and it had to be flexible to change (which our is not).

The search encompassed several of the popular systems currently on the market. We were already using two of these systems to feed our "batch" systems. We selected one of these (which happens to be the top system used in insurance agencies in the US, as well as being used in some very large companies). We made a strategic alliance with this company and now, serve each other exclusively.

A pilot project was formed with a new client. The rating system would rate and issue commercial lines policies, with data stored on the HP3000. The TurboIMAGE access (both read and write) would occur interactively. The processing of the policy would occur in the PC, thus freeing the HP3000 for other tasks (and improving response time).

We selected HP Cooperative Services for the task. Cooperative services allow a programmer to code TurboIMAGE intrinsics directly in a PC program, and with the addition of a few other intrinsics to connect to the HP3000 and server management - gives access to TurboIMAGE databases, the MPE File System, SL/XL modules and the ever-popular COMMAND intrinsic.

One problem existed, however. The rating system we were going to use was written with TurboPascal, which was not a supported language with Cooperative Services. Microsoft Pascal was, but did not provide the functionality that TurboPascal does. (Other supported languages are Microsoft C, Lattice C, Microsoft Cobol and MicroFocus Cobol.)

For TurboPascal Users, an S/R has been submitted for support as TurboPascal is the Pascal of choice in the industry, but we could not wait for the S/R to be released - we needed an answer now. The HP Response Center (specifically, with credit given to *Brian Abernathy*, who was extremely helpful - and patient), told us that we were to write our first Terminate and Stay Resident Program (TSR). These are not easy by any means.

Through the use of an extensive *CompuServe* search, we found a TSR library, which handles all of the DOS interrupts you need to handle, including pop-up, access and unload. The TSR needed to handle Connect, Server Start, Opening of two databases and then terminate (but stay resident in memory). We needed to maintain connection information which is why we needed to use a TSR. The library was initially shareware, but is now an officially supported product of Innovative Data Concepts in Horsham, PA.

The TSR uses a series of pointers to a data area which defines the database, dataset, mode, buffers and arguments - normal parameters for TurboIMAGE intrinsics. We established a user procedure which detects a function being passed (DBGET, DBFIND, DBPUT, etc). The TSR is written in "C", and uses a SWITCH/CASE statement to transfer control to the appropriate function.

We also added a function "Call_RPC", which performs a "Remote Procedure Call." The RPC is a routine resident in an XL (it can also be an SL), which is loaded and executed. The XL modules can be written so that they are unloaded immediately upon termination or stay resident in the HP3000 if they are recurrent. We use RPCs to write flat MPE files (its a lot easier than using FOPEN/FWRITE), and to perform procedures which we already use such as the use of Soundex - to keep consistency between systems and to reuse existing code.

The TSR is generic in nature, in that it will work with any database and dataset. Arguments are passed as needed. The only limitation is that the dataset buffer (record length) cannot exceed 1000 bytes. This is a limitation we placed, not a Cooperative Services limitation. If we find we need to change it - it's a two-minute change and a recompile. We looked to keep the size of the TSR as small as possible so as to keep as much PC memory free for the application.

For the rating system, we added externally called programs called interface programs. These programs are called by the TurboPascal program, passing data to them as needed. These are specialized programs which perform certain functions and can be used at any point where this function occurs.

Examples of these functions are Policy Validation (Exist/Not Exist), Alphabetic Name Search (Soundex and Chain Read functions), Policy Acquisition (multiple dataset access), Policy Update (multiple dataset write and update), Table Lookups (a generic function which accesses multiple tables using codes passed by the rating program).

The interface programs set the data area pointers needed by the TSR. When these pointers are set, a function is executed to ensure the TSR is in memory. If the TSR is not resident, an error message is returned and the user can back out gracefully. To date, this condition has not occurred - but one never knows when it will. Once it is determined that the TSR is in memory, a call is made to it, passing the data pointers and the function code (ie: DBGET, DBFIND, DBPUT, Call_RPC, etc.).

Once the function is determined, it references the data in the memory of the calling program using the pointers passed to it and executes the function. Any data to be returned is returned to the memory areas of the calling program, again, using pointers passed to it. The TurboIMAGE Status Array is returned to the calling program using a specialized array established for this purpose, as is the Cooperative Services Status Words.

The calling program can then determine if the database access, as well as the data communications was successful.

We placed code into the TSR and into interface programs to test datacomm. The Cooperative Services Status Words return results for datacomm. If suddenly, a datacomm line goes down, data is stored on the hard drive of the local file server, and the user is notified. The alternative storage facility (as we have started to call it) is used until the line comes back up. When the line comes back up, the user is again notified. After all work is done for the day, a batch procedure is run to upload all data stored locally to the HP3000, and updates the databases.

Although Cooperative Services supports both Serial and Local Area Network connections, we have elected to not support a serial connection with this system. Speed and responsiveness are key issues, which would be better served with LANs. The configuration we use with all new clients is to install a Novell Netware LAN at their site and using either 56KB or Partial T1 datacomm, and bridge that LAN into ours. We have one HP3000/950 as a node on this LAN acting as a host system and gateway to three other HP3000's linked via NS. We are using Network Services 2.1 for Netware on the PCs.

From a programmer's perspective, since access to the TSR requires at least 15 lines of code, we have created a standard function to perform this function. The programmer merely sets up pointers and values which takes five short lines and performs the function. Error handling is performed after the function returns as would normally be done (ie: checking the TurboIMAGE Condition Word, etc.).

Since we were trying to make the TSR generic, Remote Procedures (XL/SL rountines) posed a problem. The data ereas which are passed are TurboIMAGE specific. How would parameters be passed to Remote Procedures? To solve this, we decided that we would keep the same data area and reuse three areas. Remote procedures from Cooperative Services operate in a slightly different manner than a regular HP3000 call. On HP3000 programs, parameters are passed individually, whereas, through Cooperative Services, parameters which serve as "input" parameters (passed to the routine) are grouped and parameters which serve as "output" parameters are grouped. This actually proved to be useful. We use the "dataset" parameter for our procedure name (from the XL), the "arguments" buffer for input parameter values and the "dataset buffer" buffer for output parameter values.

From a runtime perspective, all of this seems to appear that response time would not be very good. To the contrary - response time from the PC is quite fast.

To illustrate, let's define the environment we are using:

- HP3000/950 with 80MB Main Memory
- Average Number of Users: 100
    - 60% Block Mode Applications (VPlus)
    - 15% Development Programming Use
    - 20% Character Mode Applications

1. Activity #1 - Simple Mode 7 DBGET against an Image Master - Instant Response.

2. Activity #2 - Chained Access against an Image Detail for 10 records, return to formatted screen - Less than one second.

3. Activity #3 - Using a Remote Procedure (stays resident), Chained Access against an Image Detail returning six entries each screen with scrolling forward and backward, 3150 record in the chain - Less than one second each screen.

4. Activity #4 - Write 10 records to an Image Detail - About 1 second.

5. Activity #5 - Write 100 records to an Image Detail - about 4.5 seconds.

This is just a sampling of activities we used to work through our tests. In every case, unless a dataset is locked for an extended duration, response seemed to be BETTER than users accessing the same databases using HP terminals on a DTC.

Phase 5 - Vision-MIS (C.01.00) - The NewWave Application):

One of the objectives of HP's NewWave Strategy is to present a consistent User Interface. This is where NewWave comes in. As we all know by now, NewWave is a Graphical User Interface, built on Microsoft Windows. It has contained within it, many outstanding features which boost a users productivity, as well as add-on products which allow a user to access data (NewWave Access), use an HP3000 (AdvanceLink, Reflections, Tymlabs/Session), Access E-Mail facilities (NewWave Mail - links directly to HPDeskManager and OpenMail on the 9000), as well as other DOS and Windows-based applications.

But the one thing that is consistent within NewWave is its consistency. The NewWave Developers Kit contains seven manuals - one of which is devoted to NewWave Standards. The menu bars and options contain the same options (ie: ACTION performs some activity and always appears first, then EDIT, etc.).

Up to NewWave 4.0, applications written for NewWave had to be written in C, using the NewWave Application Program Interfaces (API's). If anyone has written Windows code (which is complicated in itself), NewWave programs are three-times harder. NewWave 4.0 allowed the use of Dynamic Link Libraries (DLL's) to be accessible from within Agent Tasks. This meant that Agent Task Programs could now access external "C" code for specialized applications. Also with NewWave 4.0, a new object-type called the NewWave Interface allowed the creation and use of "windows", complete with edit boxes (input fields), list boxes (scrollable lists), radio and push buttons and check boxes.

The combination of Agent Tasks, Menu Tasks (like Agent Tasks, but accessed through the TASK menu bar option), NewWave Interface Windows and DLL code, means that NewWave applications can be prototyped and can actually be used within NewWave to create systems.

The first NewWave version of Vision-MIS is being written in this way. It is allowing us to prototype and make changes to the system without the heavy investment in C-literate programmers, and is allowing us to design via prototype. It is proving to be quite useful. We are also using this opportunity to standardize our presentation (user interface) and make changes which would have been difficult under the old system. Changes such as entering a security password once at the start of the application instead of in each function - which has been asked for by our clients for years. Another feature being implemented is online, context-sensitive HELP, which is a standard in NewWave, but sorely lacking in the VPlus-based system (except the message window on line 24).

There is code which we use today, which will be embodied in the NewWave Vision-MIS. Much of this is code used for specialized calculations and procedures. These will be placed in either the XL or recoded in the DLL. In most cases, to maintain consistency between the old and the new, it will be placed in the XL and called using Cooperative Services.

And as for our TSR and the programs which currently use it - these programs will be placed in the DLL with slight modifications to make them subroutines, as opposed to main programs. We have found that this generic TSR utilizes much less memory than coding for each database and provides for maximum flexibility. In addition, it provides for code which can be accessed at any time. For example, if a policy needs to be accessed, the *Get_Policy* function can be used in any module where the policy data is needed. The TSR did need some slight modifications to run and be accessed with Windows present, however, functions with the same speed as we experienced in the DOS-based rating system.

Conclusion:

The information covered in this paper only touches on our activities in this area - we could easily spend a day or two on this subject, getting down to the details. Using client/server technology will be the "wave" (pun intended) of the future. HP has clearly led the way with tools and end-user products to make this a reality.

To summarize, client/server computing in an MPE environment can start with something as simple as introducing NewWave Access to end-users, thus allowing them to access data and use tools that they are already using today. The proliferation of the PC within the workplace and home has caused an explosion in computer-literacy amongst the user community.

In addition, the user community will be looking for ways to link data from central databases to PC-based applications, whether they be word processing, spreadsheets, databases or specialized applications. NewWave has shown to provide this functionality. Even seasoned data processing professionals have embraced NewWave and Windows. These functions can be reality today in any MPE installation. Our NewWave Vision-MIS will use NewWave Write and will be able to transfer data from the application to the word processor for canned, semi-custom or custom letters to insureds and claimants.

With Cooperative Services, it is possible to use existing talent to create client/server applications. If you currently use COBOL - both Microsoft and Microfocus Cobol provide for easy migration and little change. Your database calls will require the addition of one parameter, and the program will require four new intrinsics. The most difficult part will be replacing VPlus screens, however, both vendors provide for screen formatting, which you may find easier to deal with than VPlus. In essence, you should be able to bring much of the code down to a PC and keep it intact - and it will run on a PC. This was, and still is, the objective behind Cooperative Services.

If you want "Windows" on your PC, but do not have the resources to bring the programs down to the PC now - theres VPlus for Windows. This facility translates your forms files from the HP3000 to a Windows forms file. Forms are then cached at the PC automatically, thus reducing the datacomm overhead of sending forms over the RS232 line. The beauty of this solution is that there are no changes to your HP3000 programs, and they work in tandem. This solution also makes a very good "interim" step to the client/server environment while staying in MPE.

What would be necessary in your particular installation will be based on your study of your needs. To make this switch because "it is the thing to do" is not justified in cost or manpower (unless you have the luxury of an open checkbook and many people sitting around with nothing to do).

But, if your users are demanding more in the form of information - and your backlog is high - then data access tools such as NewWave Access may be the answer.

If you have applications which may benefit from better performance executing on a PC, then a migration to this platform with interactive access to existing data may be justified. Those under-utilized PCs may then be used for more than terminal emulators - they could become instrumental as personal information systems (NewWave, using the Agent executing Access tools and formatting information, and full-fledged applications running on them accessing data on multiple platforms).

Client/Server computing for the HP3000 is here today, and it's here to stay.

Cooperative Services Examples using our TSR

Examples of Data Structures used in Interface Programs and TSR:

1. Host Data Buffer Passed to Remote Procedure Calls:

```
struct HOST_DATA {
      char  procname [16];
      int   reserved;
      char  user_data [1000];
      } pc_to_host;
```

2. TurboIMAGE Status Array used in all IMAGE Calls:

```
typedef struct {
      int  cond_word;
      int  word_2;
      long word3_4;
      long word5_6;
      long word7_8;
      long word9_10;
      }     ISTATUS_TYPE;
```

3. Example of Argument Values used for TurboIMAGE Calls:

```
typedef struct {
      int  length;
      char body[30];
      }     ARGUMENTS;
```

4. Example of Data Buffer Structure used for TurboIMAGE Calls:

```
typedef struct {
      int  length;
      char body[4096];
      }     BUFFERS;
```

5. Pointer structure used when calling the TSR for all Cooperative Services Calls:

```
typedef struct {
      unsigned int     far *hpcs_routine;
      char             far *hpcs_base;
      char             far *hpcs_dset;
      int              far *hpcs_mode;
      ISTATUS_TYPE     far *hpcs_status;
      char             far *hpcs_item;
      char             far *hpcs_list;
      BUFFERS          far *hpcs_buffer;
      ARGUMENTS        far *hpcs_argument;
      }     VISPARMSDEF;
```

Note the ISTATUS_TYPE, BUFFERS and ARGUMENTS typecast in this fragment. These are defined in the code used above and allows the use of "special" data types. In this case, the Image Status Array is a "data type".

Example of an Interface "Calling" Program to the TSR (Function "main" only):

```
void main()
{
  argument.length = 6;
  printf("\n\nEnter Agency-Branch Code (aaaabb): ");
  gets (argument.body);
  agency_br_data.length = sizeof(agency_br_data)-sizeof(int);
  /* Set up pointers to TSR */
  visparms.hpcs_routine  = (unsigned int far *) &routine;
  visparms.hpcs_dset     = (char         far *) dset;
  visparms.hpcs_list     = (char         far *) list;
  visparms.hpcs_mode     = (int          far *) &mode;
  visparms.hpcs_status   = (ISTATUS_TYPE far *) &status;
  visparms.hpcs_buffer   = (BUFFERS      far *) &agency_br_data;
  visparms.hpcs_argument = (ARGUMENTS    far *) &argument;
  UserPtr = (void far *) &visparms;
  /* Check if TSR is resident, if not, error message */
  if(TsCheckResident("VISRTL  ",&idnum) == 0xffff)
    {
    if (TsCallUserProc(idnum, UserPtr)) /*If resident, go to it */
       printf("\nUnable to call VISRTL"); /* Error */
     else
      {
    if (visparms.hpcs_status->cond_word == NO_ENTRY) /* CC = 17 */
      {
        printf("\nAgency/Branch %s  Not Found.", argument.body);
        printf("\n>%.6Fs<", agency_br_ptr->abr_agency_br_code);
       };
    if (visparms.hpcs_status->cond_word != 0)   /* DBEXPLAIN */
      DBerr("DBGET", *visparms.hpcs_status);
    else   /* Found record - Display Name of Agency */
      {
       agency_br_ptr = (struct AGENCY_BR_DEF far *)
                                 visparms.hpcs_buffer;
       MoveMemBlock((void far *) agency_br_ptr,    /*Fill Buffer*/
                    (void far *) &agency_br_data,
                    sizeof(agency_br_data));
        printf("\n%.6Fs", agency_br_ptr->abr_agency_br_code);
        printf("\n%.30Fs", agency_br_ptr->abr_agency_branch);
        printf("\n");
      };
      }
    }
  else
    printf("\nVISRTL is not loaded");
}
```

Paper Number 3354

A CLIENT-SERVER PROTOTYPE FOR IMAGE

Ross G. Hopmans
Brant Technologies Inc.
51 Tannery Street
Mississauga, Ontario
Canada L5M 1V3
(416) 858-0153

The purpose of this paper is to describe the development of a prototype application utilizing a client-server architecture. This was the first application of this technology for both the developer and the client organization and I would like to point out our design objectives, why we chose this architecture and what we learned in the process.

My premise is that there are many tools, techniques and technologies available to integrate into our existing computing environments. We can produce more effective software by matching the needs of the application with the capabilities of the tools available.

BACKGROUND

The client organization for this project was the New Brunswick Department of Advanced Education and Labour (AEL), a provincial government department responsible for community colleges, certification of trades people and all post-secondary student aid. AEL is a PowerHouse user with several TurboIMAGE data bases on their HP 3000 to maintain student records, course information, etc. and they had ambitious plans to expand the level of automation and standardization within the department.

I was brought in as a contractor to assist with the technical specifications for a computerized examination management system. This system was meant to automate a number of functions including the generation, delivery and marking of examinations. It was to accomplish this through a repository of exam questions that were categorized in a number of ways. The idea was that each course offered through the community colleges could have a defined "prescription" describing its objectives and their relative weighting in the course. If we could develop a system to automate the generation of an exam from our data base of questions according to that "prescription", we would allow the teachers to spend more time teaching and less time developing the exams. In addition, it would ensure a standardized, higher level of examinations and reduce the risk and consequences of a compromised exam.

Several years earlier, AEL had developed a host-based prototype for the "engine" to generate the examinations given a data base of questions and an exam prescription. This was a reasonably complex piece of software and it proved the viability of the idea. However, given the software tools and technology available at the time, the prototype was unsatisfactory in a number of other areas. Our design team was to determine the needs for a fielded application of this type and propose an architecture for development and delivery of the software.

VISIMAGE PC

It is worth mentioning at this point that client-server technology was already recognized within AEL as they had recently purchased a product called VISIMAGE PC. VISIMAGE is a report writer and data interchange tool for the HP 3000 which incorporates a PC front-end to the data-extraction engine on the HP.

VISIMAGE PC allows users to create and catalog their own customized reports, using a windowing environment on the PC client to speed up and simplify the development of ad-hoc reports while insulating the user from the HP 3000. It is very easy and intuitive to "point and shoot" items to use for selection and/or reporting from the scrolling menu of available data items. The user can "paint" the report on the PC screen and easily modify it for iterative development. VISIMAGE uses PPL from Walker, Richer & Quinn for its data communication mechanism.

Because VISIMAGE was a key part of AEL's future reporting strategy, client-server was, in a sense, an approved technology.

REQUIREMENTS

A number of requirements were defined for the computerized examination management system environment:

1. We wanted to maximize the investment already in place in terms of hardware, software tools, skills and standards.

2. We had to be able to accommodate an ability to link an exam question to one or more graphics. For example, in an electronics exam a circuit diagram may be an essential part of the question. The graphic had to appear with the question whether delivered on-line or in hard copy form.

3. New Brunswick is Canada's only bilingual province and so we had to be able to handle both French and English. The two languages must appear on the same screen if delivered on-line and side-by-side if delivered in hard copy format.

4. The system had to be easy to use.

Given these requirements, it was apparent that we had to move beyond the HP 3000 in order to develop our user interface. Many of the functions were very visual in nature and required a far more flexible user interaction than we could provide in a terminal-based environment on the HP.

CLIENT-SERVER

We proposed a client-server architecture using the HP 3000 as the server and PCs as clients.

Client-server applications represent the state of the art in software application technology. They make the best use of large centralized data bases while at the same time taking full advantage of the processing power available on the desktop. A client-server architecture links intelligent workstations with a large centralized computer to maximize the utility of both. The software application runs co-operatively on both hardware platforms. For our application this represented the best way to provide full functionality for the user while maintaining centralized security and control. This architecture would utilize the hundreds of mips of unused PC processing power which in turn would lighten the processing and data communications load on the central HP 3000.

Using a client-server approach, the software would present a PC-like application environment to the user. The PC would act as a client to access the services of the HP host. A client-server application is logically split so that part of the application runs on the PC (the client) and part runs on the HP host (the server). The application would be developed so that all of the data resided on the HP.

The server would run on the HP 3000 to co-ordinate the activities of the PC clients and manage the data requests. Other portions of the application, such as the exam generation module, production reports, etc., would also run on the HP 3000.

The PC client software would be developed using an application development tool running under Microsoft Windows 3.x. The development environment would combine object-oriented programming with dynamic graphics and intelligent external links to handle all interaction with the user. The PC client would perform the processing associated with screen handling, data validation, etc. This would off-load considerable activity from the HP 3000 while providing the user with all the advanced capabilities available in the PC environment. As a result, the user would experience better performance because of the dedicated processing power of the PC and because the PC can cache its data requests to the HP.

Windows development provides a consistent, intuitive interface for the user and it allows easy communication with other Windows applications. Communication between the PC client and the HP server could take place via PPL (process to process link), a development utility from Walker, Richer & Quinn. PPL was developed for just this purpose.

The user would interact with the PC client application and as a result he or she would be presented with a familiar Windows 3.x environment. Windows is already a standard in the industry and its acceptance is growing daily. Furthermore, many companies and organizations will ONLY accept Windows applications now. As a result, our application would provide all the intuitive user interface capabilities that have made Windows a success. These include pull-down menus, pop-up windows, layering of session windows, icons, integrated graphics, mouse support, easy system navigation and more. Full control over fonts, colors and images lead to very effective screens. For example, instead of answering a series of YES or NO questions with a Y or N, screens are easily developed to allow the user to click on the YES or NO button. Choices are thus clearer and no edit check is necessary. Many users are already familiar with Windows applications which would reduce training time and support requirements.

THE PROTOTYPE

The purpose of the examination management prototype was to prove that we could develop a system to meet the interface needs of the users. In other words, we had to demonstrate integrated graphics, bilingual screens and ease-of-use while accessing data on the HP 3000. We chose delivery of an on-line examination as the subset of the total fuctionality that we would prototype.

In building the prototype, we decided that we would deliver it without a keyboard. The reality is that many users are uncomfortable dealing with a computer application and probably cannot type well. However, the use of a mouse is much more intuitive. All interaction with our application would be via the mouse to point and select. Any requirement for the user to provide a numeric answer would be through clicking on a numeric template displayed on the screen. Figure 1 shows the use of the number pad window for entry of the student number.

We wanted to build a user interface that provided continuous feedback for the user and one that allowed access to all of the available functions at any time from anywhere. To accommodate that, we designed a window on the right side of the screen containing a number of buttons that the user could click on at any time. These included functions such as "Complete" to end the consultation, "Utilities" to access a calculator, the clock and other utility functions, and "Help" which explained how to use the help system. The user could right click anywhere on the screen to get context sensitive help in his language of choice at any time.

The idea was to build a PC client application that would interact with an end user intending to take an examination on-line. The user would identify himself to the PC client by his student number. The client software would then log on to the HP 3000 and pass a message to the server software saying "here is a student number". The PC software would either expect the student number to be rejected as invalid or receive back the identity of the student, his language preference and a list of exams which the student had been approved to write.

Once the PC client had received the information from the host server, it would display the student's identity and register the language preference such that all subsequent interaction with the user was in his language of choice (French, English or bilingual). The list of exams the student may write was displayed in a scrolling menu and the user could select one only. The PC client then passed that exam identifier to the host server and it expected to receive the exam questions and additional information associated with the exam.

Once the PC client had received the exam data, it dynamically re-structured the user interface to fit the data. We wanted to give the user the ability to move directly from one question on the exam to any other question at any time, as well as providing feedback on whether or not the user had answered the question. We decided to create a row of numbered buttons along the top of the exam questions window. The user could simply click on the appropriate button to go to that question. We dynamically sized the buttons to fit in the area allotted. In addition, we provided a small box directly below each button. A white box indicated that the question had not been answered and the box turned red once the question was answered.

Exam questions were downloaded with both an English and French text and multiple answer choices. In addition, a graphic file may be associated with a question. The question window had a place for the question number and text of the question, a radio button group for the answer selections and an area for the graphic. We were able to combine different foreground and background colors and fonts to enhance the presentation to the user. A radio button provided an open circle to the right of the answer text. When the user clicked near the circle of text, the open circle filled in to indicate that the selection was chosen. Only one choice was permitted and the user could easily change from one answer to another.

The questions and possible answers were displayed in the user's language of choice. If the user was bilingual, the screen split and displayed everything in both English and French. The text and graphics would dynamically resize themselves to fit the window area they were allotted. Figures 2 and 3 show examples of this interface.

Once the user indicated that they had completed the exam, the application would determine the mark by comparing the student's answers to those supplied from the HP 3000. The student was notified of his mark and the results were passed back to the host data base.

We were successful in developing a user interface that met our objectives. We could not have created as effective a user-interface on the HP 3000 alone. By offloading the user portion of the application to the PC, we were able to take advantage of the incredible flexibility and performance offered on that platform. Of course the steering committee had many suggestions as to improvements that could be incorporated into the software, but the important point is that we demonstrated that we could provide an interface that met their needs.

THE TECHNOLOGY

There were really three components to the development of the prototype: the PC client software, the communication link and the host server accessing an IMAGE data base.

The PC client was developed in Kappa-PC, an object-oriented development environment from IntelliCorp that runs under Windows 3.x. I had about 12 months' experience in Kappa, which combines true object-orientation, rule-based reasoning, dynamic graphics, links to the outside world and a procedural language. Although I had primarily used it in the development of expert systems prior to this, I found Kappa-PC to be a very productive tool for the development of any Windows application.

Kappa's OOP capabilities proved very beneficial both in our ability to create the underlying model of the application in a natural way and in our ability to bind processes to the data itself. For example, for any slot in an object (ie. field in a record) I can attach a method (ie. procedure) that will fire before or after the slot value changes. So I never have to explicitly call these procedures. The data will look after itself. The other thing that is important to understand is that everything is treated as an object and objects inherit the properties of their ancestors. The "encapsulation" of data and procedures is a strong benefit of OOP. I found it a very productive environment with its graphical development editors and browsers.

Although we would integrate a product such as WRQ's PPL in production (just as VISIMAGE PC does), we developed the prototype using MiniSoft's MS92 command language. We created a number of bat files containing MS92 commands to log on to the host, run host programs and transfer files. Kappa-PC simply executed the bat files passing the appropriate parameters. Once a file had been transferred from the host to the PC, Kappa would read in the file and process accordingly.

On the HP 3000 we had to develop a server to accept commands from the PC, access the IMAGE data base accordingly and create output in MPE files. To upload exam results, the server had to read an MPE file and update the data base. In addition, the server would format the exams for hard copy output. The server software was an easy program to develop. However, since I was not working on-site for the development of the prototype, the choice of a language was not an easy one. The client had a 955 and a 922 and I had a Micro 3000. Ironically, the HP 3000 server software for this advanced application was developed in SPL since it was the only language available on both our machines.

RESULTS

We were able to demonstrate that client-server technology was viable and could produce effective results in AEL's environment. However, we were faced with several challenges.

One was the data transfer speed. We were operating over an X.25 network at 9600 baud and that would prove to be too much of a limitation in production. One way around that limitation, and one employed by VISIMAGE, is to store some data in the PC if it is likely to be needed again in the future. In our case, exam questions and graphics are likely to be re-used by subsequent users in their exams, so we could hold those on the PC and check to see if the data was already there before downloading. Graphics tend to be very large files and it proved faster to keep them compressed on the HP 3000 and download in that format. The PC would then decompress the file.

Graphics themselves proved challenging. Most were scanned images from existing exams and very often the quality of the scanned image was insufficient to provide a suitable graphic once it had be reduced in size to fit in its screen window. Images had to be retouched with a graphics package to improve their quality.

PCs in the organization would have to be upgraded to effectively run the PC client software. Although I developed some of the application on a 286 machine with only 1 meg of memory under Windows, it was very slow. Realistically, I would recommend at least a 386 with Dos 5.0 and 4 meg of memory.

CONCLUSION

This project was important in that it demonstrated that we could develop a whole new range of applications for which our traditional development tools on the HP 3000 had proven inadequate. However, we could retain all the important benefits of using our HP 3000 for centralized storage and control.

The use of client-server technology allowed us to maximize the use of our HP 3000 as well as better utilizing the PCs in the field. Co-operative processing applications such as this take advantage of the best features on both hardware platform. By developing the user software on a PC platform with Windows, we produced a friendly environment that was easy to use and met needs that could not be addressed though a terminal-based architecture.

I feel strongly that highly graphical environments prove far more productive. In addition, the separation of client and server software can lead to lower maintenance and better re-use of software.

In conclusion, I believe that HP 3000 users can easily develop client-server software that will move your computing environment in whatever direction the future holds. No doubt there are some new tools and techniques to be learned, but you can build on your existing hardware, software and human resource skills to provide more effective applications.

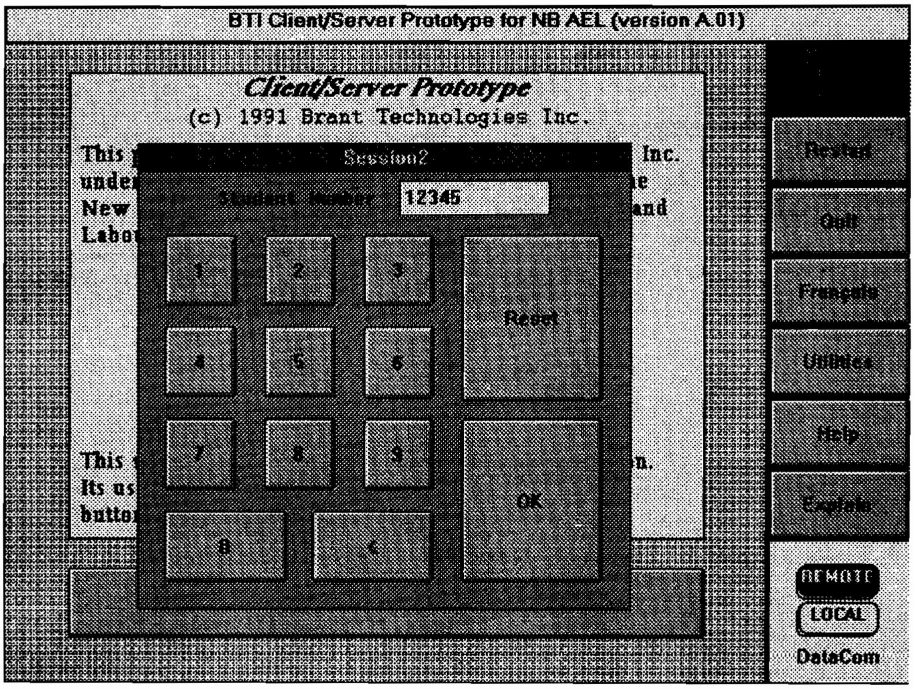
Ross G. Hopmans
1992.06.12

Kappa-PC is a trademark of IntelliCorp Inc.

VISIMAGE is a trademark of Vital Soft Inc.

PPL is a trademark of Walker Richer & Quinn Inc.

Windows is a trademark of Microsoft Inc.



BTI Client/Server Prototype for NB AEL (version A.01)

Client/Server Prototype

(c) 1991 Brant Technologies Inc.

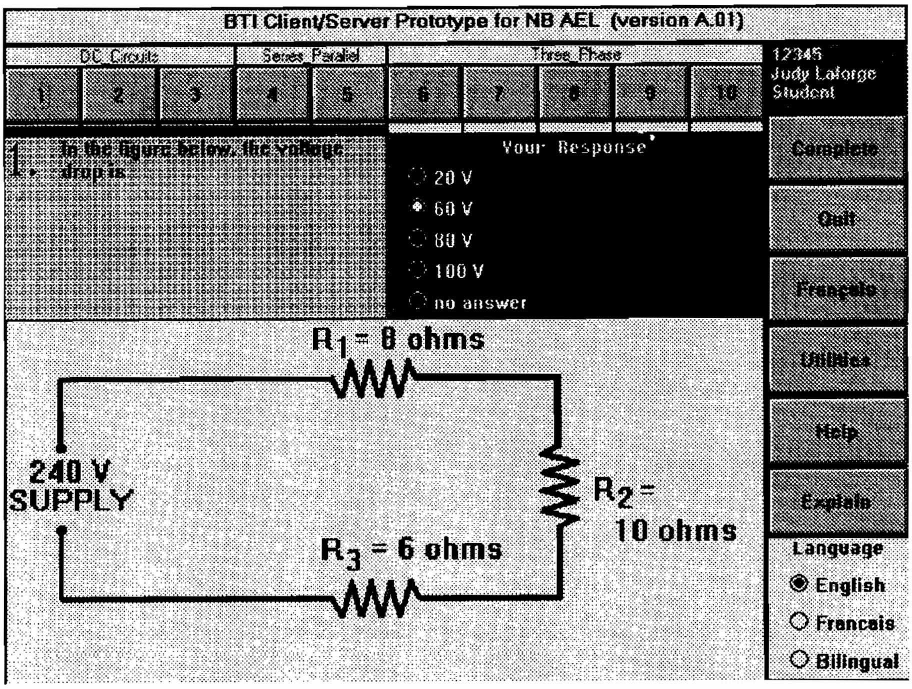
This Inc.
unde e
New and
Labo

This n.
Its us
butto

Session?

Student Number: 12345

Figure 1 shows the initial user interface once the user has passed the introductory screen. He or she must provide a student number by clicking on the number pad displayed on the screen. Note the buttons on the right that the user can access at any time.


BTI Client/Server Prototype for NB AEL (version A.01)

12345
Judy Laforge
Student

1. In the figure below, the voltage drop is:

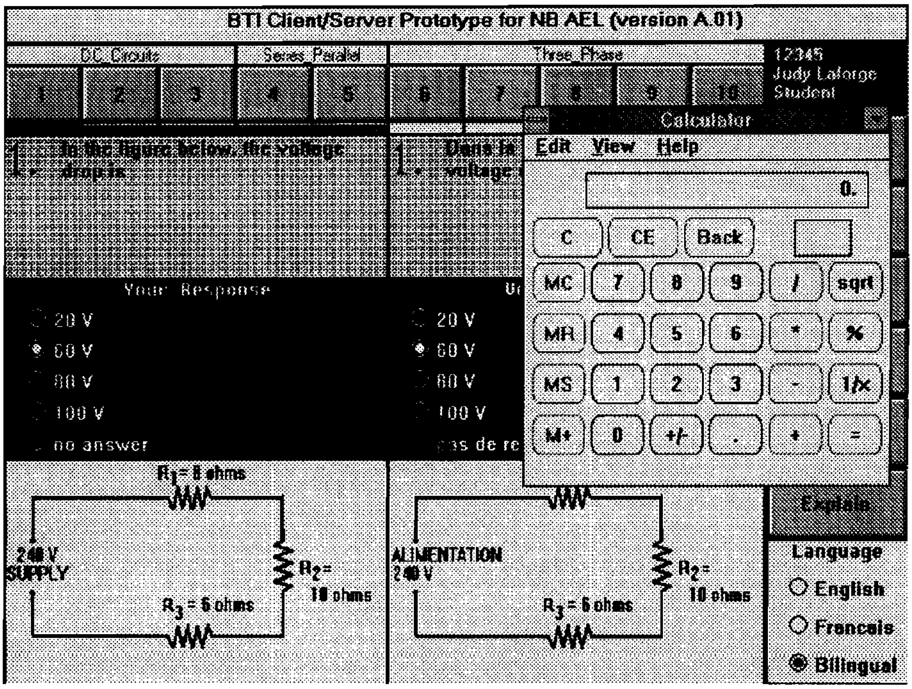
Your Response

20 V [✓ 60 V] 80 V 100 V no answer

Language: [✓ English] Francais Bilingual

In Figure 2 the user has been accepted and has chosen his or her exam to write. Note this exam has 10 questions so 10 buttons have been created to allow the user to proceed directly to any question. The small boxes below the question buttons indicate that the user has answered questions 1 through 5. Here we see question 1 in English. The text, answer choices and graphic all have their own window.

Client Server for IMAGE – **FIGURE 2**
3354 – 10



BTI Client/Server Prototype for NB AEL (version A.01)

12345
Judy Lafarge
Student

1. In the figure below, the voltage drops

1. Dans la … voltage …

Your Response

- 20 V
- [✓ 60 V]
- 80 V
- 100 V
- no answer

- 20 V
- [✓ 60 V]
- 80 V
- 100 V
- …s de re…

Language
- English
- Francais
- [✓ Bilingual]

Figure 3 is similar to Figure 2 except the user has switched to bilingual mode and so we see French and English side by side. In addition, the user has selected the calculator from the "Utilities" button.

Paper Number 3355
Shop Floor Data Collection

Kirk E. Bedwell
Marsh Company
707 East B Street
Belleville, IL 62221
(618) 234-1122

Overview

Shop floor data collection includes the collecting of payroll and labor information. Collecting can be done by either personal computers or HP 3000 series computers. Whatever the mode of data collection, it is imperative that data be collected accurately.

This paper will concentrate on the collection of data using a personal computer network. Both payroll and labor data collection will be discussed. Topics including choosing the proper collection platform, what type of data is collected and various rules for collecting the data are discussed.

PC Data Collection Versus HP 3000 Data Collection

There are a number of factors to consider when deciding which platform to collect shop floor data. Although most applications may already reside in the HP 3000, it may be advantageous to collect data on PC network.

The first factor to consider is the number of employees that will be clocking in and out on payroll and time and attendance. It is imperative to keep the flow of people using the shop floor clocks moving as fast as possible. If the clocks are posting their transactions to a HP 3000, the 3000 program controlling these clocks should be receiving the highest possible priority. If 500 people are clocking in and out, the responsiveness of the 3000 to other users may suffer during these times. If a personal computer is used for shop floor data collection, then the only operation that computer is performing is the actual updating of the employee punches.

A disadvantage of using the personal computer is that employee numbers, work order numbers and other information must be downloaded for validation. Validation is the process where an employee swipes a badge or wands a bar code and the information read by the clock is approved. The other disadvantage is that all of the punch information must be then uploaded to the host HP 3000 computer. In todays market, however, the transfer of files between computers is a minor hurdle which must be cleared.

Payroll Data Collection

Payroll information to be collected at the time clock includes an employee badge number and the time of the punch. The payroll system should then be able to calculate the number of hours worked, calculate overtime and track vacation, sick and other times accumulated by each employee.

The payroll data collection should have interface capabilities to the company payroll system whether it be a packaged program, service bureau or internally developed system.

Labor Data Collection

Labor information to be collected at the time clock includes the employee badge, work center, work order, start time, end time, quantity completed, quantity scraped and scrap reason code. By collecting this information on the shop floor, manual data entry can be eliminated. The installation of a pc network to collect this information frees the HP 3000 from constantly monitoring the job clocks. However, provisions must be made to allow the transfer of data files between the 3000 and the pc network.

The key issue in automating shop floor data collection is allowing the proper decision maker to have access to the information retrieved from the shop floor. By transferring the information to the HP 3000 from the shop floor, all users can be granted access to the data.

Data Transmission Between Platforms

Each shop is different in regards to the timing of data transmissions between the shop floor and the host computer systems. A factor to consider in the number of file transfers is the nature of the business. For example, in a manufacturing firm running a Just In Type (JCT) inventory, it may be critical to transfer information several times an hour. Other firms may be able to utilize an hourly or even longer transmission period. Whatever transmission times are selected, all users must be aware of these times.

Winning Strategies for PC Managers
Paper 3356

Brad Feazell
Manager of Information Systems
Dril-Quip, Inc.
13550 Hempstead Highway
Houston, TX 77040
(713) 939-7711

A New Challenge

With Personal Computers celebrating only their eleventh birthday this year, the challenge of PC management is relatively new. Also relatively new, are the duties of PC managers, coordinators and support people. These responsibilities are sometimes assumed by official computer people and other times consigned to the most capable and available person.

The IBM compatible PC has developed into a quite capable and complex machine. My first computer had only 5k of RAM. I could plug the keyboard into a television set and cassette tape recorder and I had all the PC I needed. Today, PC configurations with 4-16 MB RAM and 80-300 MB of hard disk storage are common and the array of available software and hardware peripherals is mind boggling.

The 1980s saw an explosion of Personal Computer use that has redefined the way many workers do their jobs. Some still argue that PCs have not increased productivity, but I challenge them to consider the productivity of modern accountants without spreadsheets or secretaries without word processors.

Traditionally, corporate use of PCs was to limited to departmental word processing, number crunching, desktop publishing and database tasks. Today we are finding PCs at the top of the corporate computer chain, carrying out mission critical tasks and processing strategic corporate information.

The Role of MIS

MIS role in managing PCs varies from company to company. When PCs first came on the corporate scene, there were no rules for managing them. Was it a glorified typewriter or a real computer? Was it to be purchased and maintained like a typewriter or should it be given real computer status? Who do you call when something goes wrong? There are several approaches to managing corporate PCs.

Approaches to PC Management

First, there is the departmental approach. Each department is independently responsible for purchasing, maintaining and supporting their own PCs. Users have a great deal of freedom in making hardware and software choices. This approach worked pretty well when most PCs were being used for Wordstar and Lotus 1-2-3 version 1A. Today, there are some obvious problems with this method. When companies using this approach install LANs, the smorgasbord of incompatible hardware and software that usually accumulates creates a connectivity nightmare. When you compound these connectivity problems with today's more complex software, computing starts to interfere with the primary purpose of the department.

Second, there is the heavy handed MIS "control" approach where MIS purchases and maintains all PCs. Mainframe techniques are applied to managing PCs and the user has little or no choice in the selection of hardware or software. The problem with this approach is that MIS mainframe practices just don't work with PCs.

Traditionally, MIS has been slow and methodical in response to user requests, (which are usually for computer programs). In the development phase, systems are analyzed and flowcharted. Then programs are coded and documented (sometimes) and presented to the user in completed form. The user responds, "This is not what I asked for" and MIS goes into the maintenance phase. Eventually, the user has a usable program. The use of PCs just does not fit into this kind of methodology and quite frankly, neither does mainframe program development.

Last but not least, there is a partnership approach where MIS is responsible for the big picture—the long term health and direction of the computing environment as a whole. Users are partners, not recipients of mandated policy and procedure. MIS is responsible, yet receptive and adaptable. Users play a part in decision making processes. This is scary for some of you isn't it. This approach may be scary and it may make us feel more insecure, but it works and many companies are finding that it is the best way to harness the computing power that sits on users' desks throughout the organization.

Lack of MIS Enthusiasm

Many MIS people are less than enthusiastic about PCs. They do not consider PCs industrial strength computers and of course, early on they were not. We now have inexpensive PCs that can exceed the raw computing power of some mainframes. PCs are replacing minis and mainframes in companies of all sizes.

You may say, "DOS can't come close to the capabilities of mini and mainframe operating systems" or "Data reliability and disk I/O is not up to par on PCs," and you may be right. However, this is not stopping the trend toward PCs. So why would we sacrifice the greater functionality, reliability and speed of mainframes to move corporate computing to Personal Computers? There are several reasons, not the least of which is cost. Have you looked at the cost of high end PCs lately? Complete 50 MHz 486 systems with 4 Mb RAM and over 200 MB of hard disk storage are available for less than $2,500. PC software applications, databases and compilers are also "orders of magnitude" cheaper than their mainframe counterparts.

Another reason is that there is also functionality available on PCs that is not available (or at least not as robust) on mainframes. I am speaking of improved relational databases, compilers, graphical user interfaces, electronic mail, groupware, desktop publishing, etc.

Little time has passed since the early days of PCs, but much has changed. Hopefully, we too, have changed. Perhaps you are one of the few who embraced personal computers early on. Either way, we have come full swing into the new age of personal computing. How we manage this resource can profoundly affect the companies we work for.

No Small Task—A Comparison of PC and Terminal Users

A Technical Support Nightmare

From the perspective of technical support, effectively supporting 50 PCs can require as much effort as effectively supporting 100, 200 or even more mainframe terminal users. Why is PC support so much more difficult? Lets look at some comparisons of PC and terminal users.

On a mainframe, terminal users can not change the system's configuration or reboot the machine. By contrast, the configuration of the PC can be changed so easily by the user that it's next to impossible to protect it. On a mainframe, a file can be given a "read only" attribute that can not be changed by un-authorized users. On stand-alone PCs, "read only" attributes can be easily removed by any user. However, many PC network operating systems have very good file security provisions.

We do not usually fear that terminal users will damage the mainframe. PC users however, can quickly and completely obliterate the contents of their hard disk if they choose to. The user can actually erase the entire operating system with one command! No questions asked—I have deleted the operating system for you—I am at your service: C:\>.

Many PC users are quite aware of the PC's capabilities. On the other hand, most terminal users can be easily snowed by computer people talking mainframe techno-babble they don't understand.

Personal Computers can Empower Highly Motivated Workers

Progressive, highly motivated workers like PCs because PCs make them more productive. They use them for scheduling their time, storing their miscellaneous information, comparing figures and making their ideas look great on paper, all without the intervention of computer people to slow things down. PCs are about freedom and empowering.

Where Do We Go From Here?

As a new problem, effectively managing PCs does not have a textbook method for success. One thing is certain: New thinking must prevail, but now more than ever, control and standardization are paramount. Control and standardization are "fightin" words to many PC users. How do we achieve control and standardization without mutiny? Well, that is the subject of the rest of this paper.

Why Control Personal Computers?

After all, personal computers are supposed to be personal—right? Absolutely! Our goal should never be to remove the user's freedom, or their potential for creativity. That is a battle we would certainly lose.

While many PC users are becoming knowledgeable and capable enough to make effective use of their equipment, the more technical aspects of software and networking are keeping far ahead of most. If we do not establish a predictable and well-standardized PC environment, the state-of-the-art in our PC shop will be—chaos.

In corporate desktop computing, "control" can usually be translated into "standards." So how can we apply controls in a way that will preserve the user's freedom that is necessary for productive computer use?

Standardization is the Key

Why standards? We know that standards work well for writing program code and naming files. But what can standards do for us in managing PCs?

The well-applied use of standards yields interchangeability, and just as the advent of interchangeability triggered the industrial revolution, interchangeability can revolutionize your PC environment. This concept works as well for software as is does for hardware. When you follow standards in the purchase and application of your PC hardware and software, your interchangeability is increased. A well-standardized environment is predictable, easier to maintain and facilitates the implementation of new technologies. Only companies that standardize will truly be positioned for computing in the '90s.

You will probably never hear of a large successful implementation of today's more advanced environments like Windows, OS/2 or New Wave, without a high level of standardization.

Traits and Practices of a Successful PC Manager

Aside from keeping the machinery in working order, the successful PC Manager must be thoroughly knowledgeable in matters of hardware, software and networking. Available technologies must be continually appraised and appropriate technologies implemented. How does one maintain the level of knowledge required to stay on top of an industry that moves as fast as the PC industry?

1. Read A Lot; Read the Right Stuff.

When you chose to work in this field, you chose to remain a student for as long as you stay in it. A lot of reading comes standard with the job. However, the successful PC manager will be selective about his reading material. There is a glut of "industry watch" publications. News about upcoming products, mergers and high level personnel changes in top technology firms are the sort of subjects covered. Read just enough of this material to keep yourself abreast of announcements that pertain to your environment. I find Computer World and PC Week (both weekly publications) quite enough to keep abreast of up and coming PC related announcements. For networking, Network World is also very good.

Plenty of technical, hands-on type reading must also be done. This helps to offset the "industry watch" reading with some reality. PC Computing is a good practical hands-on publication. This monthly publication sometimes takes me an entire week to read even when dedicating up to an hour a day. My wife would like to cancel this subscription.

It can be difficult to keep up with all the publications that come in. It helps to learn to scan rapidly for *key* words that apply to your environment. Your key words will be different from mine. My eyes stop on words like HP, Dell, 3000, 9000, Netware, Sun, Windows ... you get the picture. It is never necessary to read every word of a magazine. Once you select your reading sources, never throw away a single issue without at least scanning for key words. You could miss an important solution to one of your problems or an announcement that will pull a long-term project together.

2. Scrutinize Your Hardware and Software Purchases.

As a manager, your hardware and software decisions are some of the most important and risky decisions you will make. These decisions can actually make or break you. So how do you maintain a winning average in selecting hardware and software? Refer to #1.

In addition, here are a couple of good rules of thumb to follow on software purchases:

- Never buy version *x*.0 of anything without a giving it a thorough checkout. There was a time just a few years ago when the decision to acquire program upgrades was almost automatic. Today, too much software is being sold today that should still be in beta or even alpha testing.

- Do not put off purchase decisions based on or merger announcements. Some of these announcements never materialize and those that do rarely have a short term impact. Consider mergers worth noting when you can buy the resulting products.

3. Know your Software Products.

When new PC products or upgrades to old ones come in, take time to become familiar with the program's new functionality. If you are responsible for providing the foresight for your environment, then you will need a solid understanding of your environment's capabilities. This does not mean that you have to become an expert user of all new products, but you should be knowledgeable in how and where they can be applied.

4. Limit Your Hardware Selection.

When selecting PCs or add-on cards, especially network adapters, limit your selection to one or two vendors. Conflicts in IRQ and memory addressing are common and the best way to get control of these problems is to standardize.

Be aware that not all PCs made by the same vendor are equal in terms of hardware and software compatibility. I have found certain hardware and software configurations that work on every HP Vectra model we have except the QS/16s. Don't assume that a setup will work until you've tested it on each machine.

Never buy an 80286 machine.

5. Use Your Network to Reduce Your Workload

If you have a network, make it work for you. Providing that you have available bandwidth and disk space on your network, there is a couple of simple techniques that you can use to make your life easier.

- Store *all* programs and user data on the file server. Doing this can reduce or even eliminate the need for local hard drives. The less you store on the user's hard disks, the less traveling is required for troubleshooting and routine updates.

- As a part of each user's network login sequence, copy the CONFIG.SYS and AUTOEXEC.BAT files from the local hard drive to the user's private data area on the file server. This way, if a user has a problem, you can check his configuration without leaving your desk. Additional benefits will be easier configuration documentation and monitoring of unauthorized changes to configurations.

Many network administrators who are using only five percent of their available bandwidth, still hesitate to intentionally increase traffic on the network. They automatically blame poor network performance on too much traffic. It takes much more traffic than most people think to overload a ten megabit per second network. So if you have the capacity, use it to your advantage. If by chance you do overload the network, look into bridges or routers to solve the problem.

Tip for Windows users: If you have a shared copy of Windows on your file server, you can perform the workstation installation (setup/n) to the users private data area on the server instead of the user's hard disk. When you have many Windows users, this enhances your ability to support Windows by giving you easy access to their initialization and configuration files. It also allows users to logon to workstations other than their own and still use their own personalized desktop settings.

6. Enlist the Help of Expert Users To Field Software Questions

There are probably some expert users who are very knowledgeable about at least one application in your shop. You can have a win-win situation if you enlist their help as a un-official support person.

Most skilled users enjoy having others ask them for help on software. They get a chance to show their stuff and be considered a "computer expert." At the same time, you are reducing your workload by re-directing calls to the resident expert.

7. Become a Configuration Expert.

One of the more mysterious aspects of PCs is what goes into the configuration files, CONFIG.SYS and AUTOEXEC.BAT. A good PC manager is intimate with the contents and options of these files. If you set up these files well, your machine will hum. If you set them up poorly, your machine may crash every few minutes or not boot at all. When you're having a software problem and you call technical support, they usually want to know the contents of your configuration files because many problems can be traced to incorrect

Standardize on a DOS Version.

Configuration files can change drastically from one DOS version to the next. At this writing, DOS 5.0 is the most recent version and it has passed *most* of my reliability testing. WARNING: Some DOS programs become unstable with DOS loaded high (a new feature of DOS 5.0). If you are loading DOS high and can't figure out why Quattro Pro or WordPerfect hangs intermittently, try removing the DOS=HIGH line from CONFIG.SYS.

Standardize Your Configuration Files.

Thoroughly test a good standard configuration and implement it company-wide. There will be some differences between various hardware and software setups, so keep standards for each configuration.

Use a Third Party Memory Manager.

If you are loading network drivers or other device drivers and TSRs, you will benefit from a third party memory manager like QEMM386. Without getting too technical, this type of memory manager will increase your available memory by using memory addresses that DOS normally does not have access to.

Windows runs better with plenty of memory and QEMM386 can help if you are low on memory. Many people think that Windows has broken the 640k barrier; this is mostly hogwash. If you're low on conventional RAM, Windows becomes sluggish and unreliable. If only 420k of conventional RAM is available before starting Windows, DOS programs running under Windows will have slightly less than 420k to run in. Most PC programs in existence today still require or run better with plenty of available conventional RAM.

8. Be Thick Skinned

Be warned that there will be problems when you are running a progressive PC shop. If you are not experiencing some perplexing problems in the management of your PCs today, then you and your users are sitting still. In this new world of multi-tasking on a single-tasking operating system, compatibility problems and memory conflicts abound. Be willing to test each new configuration extensively (this will probably not be practical unless you have standardized).

Some users can be real "whiners." Don't take their criticism personally. They have a job to do. If you introduce a new configuration to their PC that leaves them inoperable, give them your full attention until they are operational again.

9. Always Leave Yourself an Out.

I never cease to be amazed at most PC technician's willingness to overhaul a user's PC without backing it up. How do you respond to an irate user whose hard earned data you have just destroyed? I choose not to. Always back up when there is *any* risk to the user's data. It doesn't take many moaning users to deteriorate a good MIS department's image.

10. Discern Trivia from Important Issues.

You must be able to distinguish trivia from things that truly affect your environment. Consider an example: One of your PC users has installed a clever new screen saver program on his PC that is not the "company authorized" program but is clearly more impressive. On one hand, this user has probably broken several well-intended rules regarding software installation on company PCs. On the other hand, the user is quite proud of the new program and his coworkers are really impressed. What do you do:

A) Inform the user that he has broken policy and insist that he remove the program.
B) Quietly and discretely remove the program at your next available opportunity.
C) Install the program on all other PCs.
D) Mentally note the situation and keep a watchful eye on the area for further developments of more significance. Look for an opportunity to incorporate the program into the company standard or supersede it with a more functional product.

Each company has a unique personality and political atmosphere so naturally no one answer is the same for all, but here are some guidelines. In this case I would go with D (big surprise). If the political atmosphere in the company is right, I might go with B. After that, if the program shows up again I may have to go with A. Here's the point: **Be flexible when dealing with trivia**, you cannot afford to develop a "mean ol' MIS" image. Save your big guns for issues that really threaten your environment and you'll avoid alienating your users.

11. Do not Hesitate to Call Technical Support.

When a problem comes up that requires your attention, you are obligated to investigate and search the product's documentation before calling the vendor's technical support. You expect the same from your users don't you? Once you have done so and still have no solution, don't waste time banging your head against the wall. How many times have you said, "I'll make just one more change, if it doesn't work, I'll call." It's a lot like refusing to stop and ask for directions when you're lost. Swallow your pride and make the call; technical support is a service you're entitled to. Once you have made the call, you can be more productive while they figure out your problem.

Consider using CompuServe for posting technical support questions. The access rates for CompuServe memberships have recently been decreased and questions are answered promptly by most vendors.

12. Establish a Company Policy Prohibiting Software Piracy.

The reasons to take action against software piracy are not strictly moral or ethical, although the moral and ethical issues should be quite enough. We find ourselves in a time when very few people consider copying software to be wrong. In fact, in some companies, it is normal operating procedure to purchase only one legitimate copy of each software package and distribute it to users with books acquired from book stores.

There is a very real legal consequence to this sort of practice. It is called the SPA, the Software Publishers Association. According to a conservative estimate by the SPA, the PC software industry is losing $2.4 billion per year to pirates. In recent years, the SPA has ramped up its investigation and prosecution of companies who pirate software. From time to time, the trade rags run a few stories about larger companies who are being caught and prosecuted. However, the SPA targets small to medium sized companies because these companies they say, are where most of the pirating takes place. The SPA says that disgruntled employees calling in to SPA anti-piracy hot lines are frequently the source of their tips.

In addition to the threat of prosecution by the SPA, there is also a matter of programmers not being paid for their work. Can you imagine how much cheaper PC software could be if every copy of PC software in use today was bought and paid for? Think of how much more compensation would be available to pay programmers and to spend on research and developement. The system is broke folks and we, the users, are losing because of it.

Pulling It All Together.

Take a Good Physical Inventory

Lets face it, computer inventory is dull. The last thing most of us want to spend our valuable time on is taking inventory of our equipment. However, there are great benefits to maintaining a good inventory. Once the inventory is accurate and complete, keeping it that way is a snap. Getting it accurate and complete is the tough part.

A Real Life Example

When I first realized that I was spending far too much time and effort responding to users whose PCs were frozen or had some sort of error, I became persuaded that a consistent environment would reduce these problems. Or at least if one user had a problem, other users with the same configuration should be experiencing the same problem. So if all of these consistently configured users were having the same problem, I would only have to find the solution one time.

I began to formulate a plan to bring all of our PCs up to a consistent and reliable configuration. The first logical step was to test configurations until I found a good reliable setup for each different type of machine. The testing process included finding unique IRQ settings for mouse, network and other cards that did not conflict with anything else in the machine. For network cards, I also found good address settings for RAM base and I/O base. On machines that had expanded memory cards, I worked out a good mixture of expanded and extended memory for running our software. I also created consistent CONFIG.SYS and AUTOEXEC.BAT files that worked well with the new hardware setups.

The second step was to take a physical inventory of each PC and implement the new configurations wherever deviations were found. During the inventory process, a detailed list of each PC's characteristics was compiled. This included many details about the PC, even the slot number of each board on the PC bus as well as dip switch and jumper settings.

I set a goal to inventory one PC each day. Some machines were in really bad shape and took several hours to finish. However, as it turned out, I averaged two PCs per day and the task was not as difficult or time consuming as I had imagined.

During the inventory process, at about the halfway mark, I noticed something very strange about the users whose PCs I had already finished with—they were silent! Their machines had stopped freezing, performance was good, error messages had vanished. Now I was really motivated! The theory was working.

Now I can truly say, the effort was well worth it. The time and effort required to respond to PC related problems had been diminished by 50%-75%.

Conclusion

Of course it would be difficult to include *all* the requirements for successful PC management in one short paper. The ideas presented here have resulted from years of hard work, experimentation and numerous mistakes in various PC environments. While these concepts are working well for us at Dril-Quip, Inc., remember that there are sometimes political, economic and other obstacles that limit the impact of any well-designed plan.

Having introduced that negative note, let me assure you that any company desiring to take advantage of new technologies like client/server computing, will realize the importance of standardization. If the barriers in your organization seem impenetrable, I encourage you to start small, employing one or two of these methods as you have opportunity. You will probably find that the natural progression of the industry will follow you (or better, you will be following it) and doors will start to open.

Paper #:3357

The HP3000 and PC Application Software...Together at Last!

Patricia Irene Loo
Idaho National Engineering Laboratory
EG&G Idaho, Inc.
P.O. Box 1625
Idaho Falls, Idaho 83415-3405
(208) 526-6063

1.0 Introduction

Do you hear comments such as, "Entering text is so cumbersome, can't you provide us with a decent word processor like WordPerfect or Microsoft Word"? Do your applications require extensive narrative entry? Do your users use HPDesk and detest the "editor"? Is printing a hardcopy of a report a "real chore" for your Reflection users? If you answered "yes" to any of the previous questions, a possible solution to your problem(s) can be found in this review of how the Management Information Systems (MIS) Unit at EG&G Idaho utilized the personal computer (PC) to make the HP3000 more "user-friendly". The following sections describe various applications available on the Safety Performance Measurement System (SPMS) and the Occurrence Reporting and Processing System (ORPS) that utilize state-of-the-art methods to integrate PC applications with HP3000 applications. SPMS and ORPS are operated by the MIS Unit of the System Safety Development Center (SSDC) at EG&G Idaho, Inc.

2.0 Incorporating PC Applications within an HP3000 Application

In order to incorporate PC applications within an HP3000 program two software packages are required. Reflection by Walker, Richer, and Quinn and another third-party software package called "WordPerfect Toolkit" by MiniSoft. To use this software the user must execute a batch file from the PC which runs a MiniSoft program. (This program loads Reflection and moves it to background during the file transfer process). For the transfer process and loading of the PC application, a script file located on the HP3000 is read to determine the file to transfer and what PC application to execute (i.e., WordPerfect, MS Word, Brief). Figure 1 shows a script file that loads WordPerfect.

```
1
0 \RTransferring the text to the PC, please wait...
1 \wptkdoc! %0 A
9 wp.exe /m-tktxtin (modify to load a different PC application)
4 purge %0
0 \RTransferring the edited text to the Host, please wait...
2 \wptkdoc! %0 A 132
0 done\R\EK
3 del \wptkdoc!
7
```

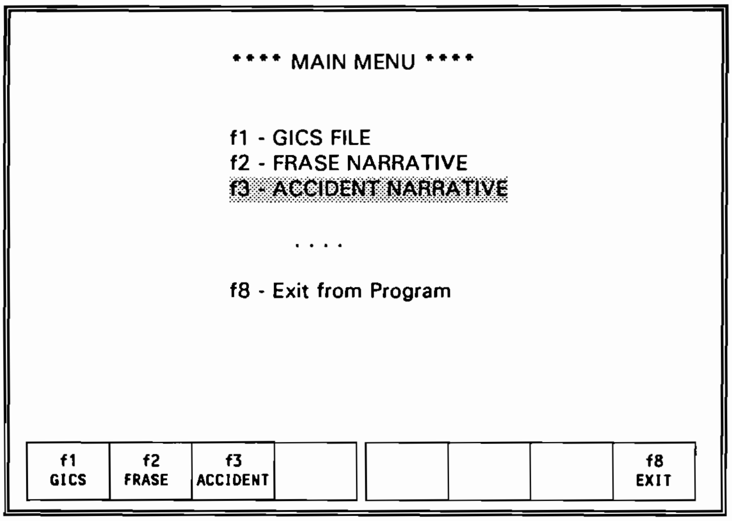
Figure 1. Example script file used to load WordPerfect.

Note: For this process to execute properly, it is important that the user's Reflection software be configured correctly for transfers and that a reasonable amount of PC executable memory be available (approx. 200K).

The following sections describe two applications MIS has placed into production using this process and summarizes the results.

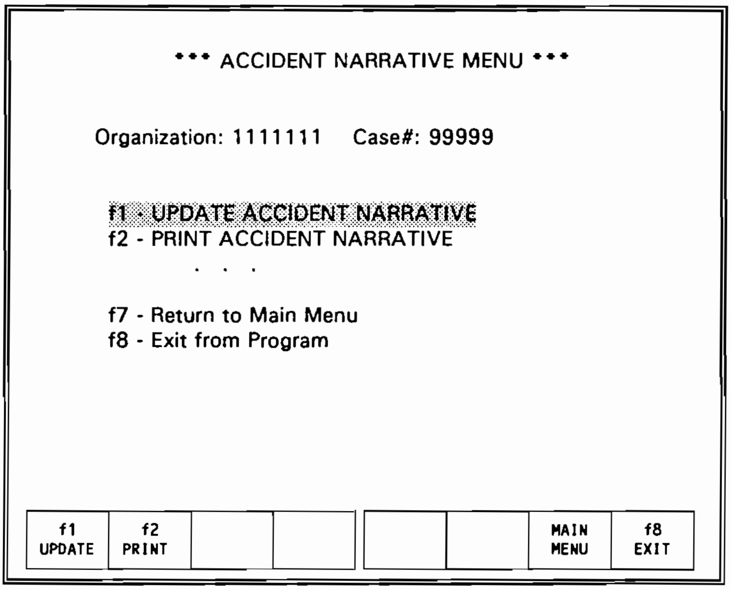
2.1 Loading WordPerfect from an HP3000 VPLUS Application

The Computerized Accident/Incident Reporting System (CAIRS) located on the SPMS requires DOE accident narrative data entry by MIS personnel on a daily basis. This program loads WordPerfect from within a VPLUS data entry application for narrative entry. After WordPerfect is loaded, a WordPerfect macro is executed which sets the font to a fixed font, sets the left and right margins, and loads the desired file. The following shows an example "run-through" of this application.



```
                **** MAIN MENU ****

               f1 - GICS FILE
               f2 - FRASE NARRATIVE
               f3 - ACCIDENT NARRATIVE

                    . . . .

               f8 - Exit from Program
```

Press f3


*** ACCIDENT NARRATIVE MENU ***

Organization: 1111111   Case#: 99999

f1 - UPDATE ACCIDENT NARRATIVE
f2 - PRINT ACCIDENT NARRATIVE

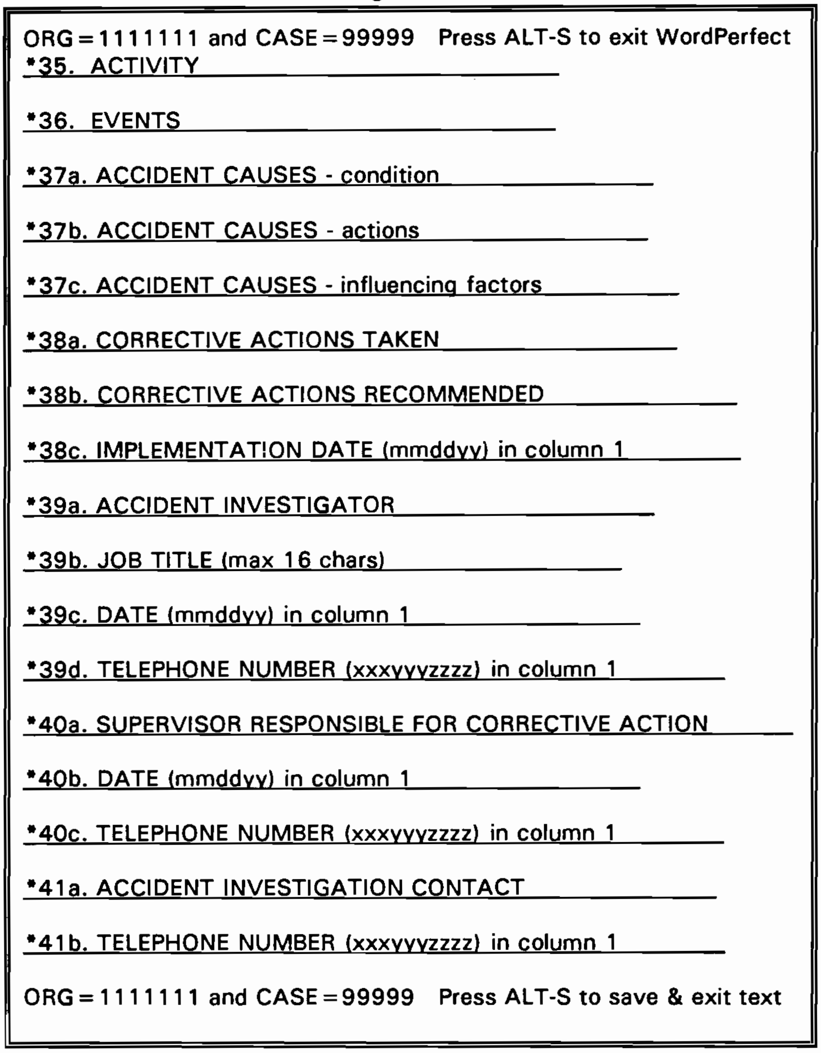
. . .

f7 - Return to Main Menu
f8 - Exit from Program

Press f1

(Transfers file to PC and loads WordPerfect.)

(Note: The following is a WordPerfect screen.)



```
ORG=1111111 and CASE=99999   Press ALT-S to exit WordPerfect
*35.  ACTIVITY

*36.  EVENTS

*37a. ACCIDENT CAUSES - condition

*37b. ACCIDENT CAUSES - actions

*37c. ACCIDENT CAUSES - influencing factors

*38a. CORRECTIVE ACTIONS TAKEN

*38b. CORRECTIVE ACTIONS RECOMMENDED

*38c. IMPLEMENTATION DATE (mmddyy) in column 1

*39a. ACCIDENT INVESTIGATOR

*39b. JOB TITLE (max 16 chars)

*39c. DATE (mmddyy) in column 1

*39d. TELEPHONE NUMBER (xxxyyyzzzz) in column 1

*40a. SUPERVISOR RESPONSIBLE FOR CORRECTIVE ACTION

*40b. DATE (mmddyy) in column 1

*40c. TELEPHONE NUMBER (xxxyyyzzzz) in column 1

*41a. ACCIDENT INVESTIGATION CONTACT

*41b. TELEPHONE NUMBER (xxxyyyzzzz) in column 1

ORG=1111111 and CASE=99999   Press ALT-S to save & exit text
```

Enter text

(Note: The following is a WordPerfect screen.)

ORG=1111111 and CASE=99999 Press ALT-S to exit WordPerfect

*35. ACTIVITY

Employee was returning from a work assignment.

*36. EVENTS

Employee returned to the 241-24H HP office with the van. The employee placed the control lever in park and walked toward the office. The van rolled backwards about 75 ft. striking a construction power truck. No injuries were incurred but the van was damaged. No visible damage to the power truck.

*37a. ACCIDENT CAUSES - condition

The control lever was found in reverse after the accident. The emergency brake was not set.

*37b. ACCIDENT CAUSES - actions

Employee exited the vehicle without setting the parking brake.

*37c. ACCIDENT CAUSES - influencing factors

The Jalope Motor Company has issued a recall on the Jalope Vans because of the transmission not staying in the park position. Procedures were adequate.

*38a. CORRECTIVE ACTIONS TAKEN

1. A corrective contact was issued to employee for not following the Safety Manuals rules for operating a vehicle.
2. A memo was sent to employee informing him of the problem with the vans.
3. A memo was issued to everyone in HP warning them of the problems with the vans.

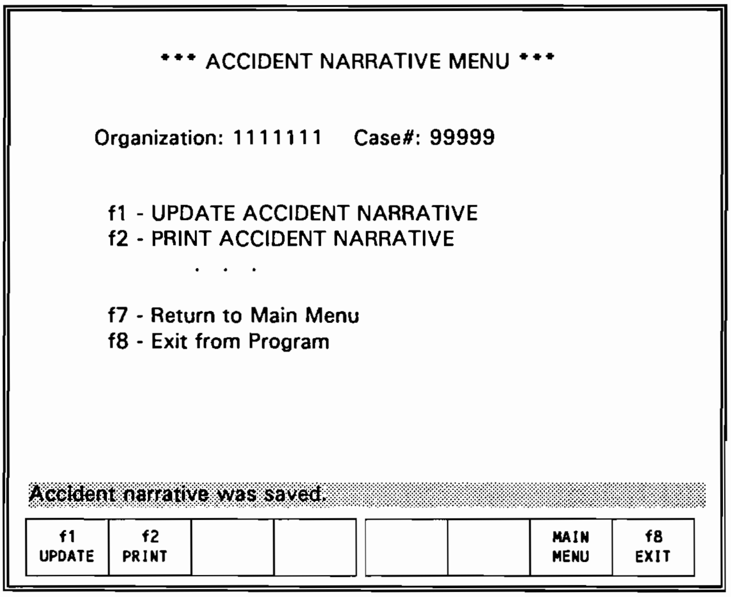
*38b. CORRECTIVE ACTIONS RECOMMENDED

We are preparing an informative contract for the HP Dept. pointing out the procedure and some safety information concerning vehicle operation.

(...continues)

Press Alt-S to Save

(Transfers file to HP3000 and returns to HP3000 application.)


*** ACCIDENT NARRATIVE MENU ***

Organization: 1111111   Case#: 99999

f1 - UPDATE ACCIDENT NARRATIVE
f2 - PRINT ACCIDENT NARRATIVE
. . .

f7 - Return to Main Menu
f8 - Exit from Program

Accident narrative was saved.

2.2 Loading BRIEF from an HP3000 Application

Other PC applications, such as BRIEF (a text editor), also can be integrated into an HP3000 program using the same procedure described earlier. The only change required is in the script file. The line that loads the PC application must be modified to load BRIEF. An example script file is shown below.

```
1
0 \RTransferring the text to the PC, please wait...
1 \tktxtin!.txt %0 A
9 c:\brief\b.exe tktxtin!.txt
4 purge %0
0 \RTransferring the edited text to the Host, please wait...
2 \tktxtin!.txt %0 A 132
0 done\R\EK
3 del \tktxtin!.txt
7
```

Figure 2. Example script file used to load BRIEF.

The following flow sequence shows an example of an HP3000 program that calls BRIEF.

```
Enter Assessment ID: 000001
Enter Media (AI, TM, etc.): AI

Transferring text to the PC, please wait....
```

(Transfers file to PC and loads BRIEF)

(Note: The following is a BRIEF screen)

```
Assessment ID: 000001
~AI00 0001
  This is the narrative for finding number 1 with performance
objective AI00. This finding expresses what information was
found regarding this performance objective. There may be
many findings for a particular performance objective...
~AI00 0002
  This is the narrative for finding number 2 with performance
objective AI00.
~AI00 0003
  This is the narrative for finding number 3 with performance
objective AI00.
```

Save text and exit

(Transfers file to HP3000 and returns to HP3000 application.)

```
Transferring the edited text to the host, please wait...

We are ready to modify database...
Do you want to save changes (Y/n)? [Y]Y

Do you want to quit (y/N)? [N]Y
```

2.3 Summary of Integrating a PC Application into an HP3000 Application

As can be seen from the previous two applications, it is quite simple to integrate various PC applications into your HP3000 program so that the users see the process as one application. There is some delay during the transfer and load process; however, with state-of-the-art equipment this delay can be kept to a minimum. The delay time is comparable to loading many HP3000 programs from an application and is dependent upon the file size, file transfer rate, and CPU speed. The primary difficulty in using the WordPerfect Toolkit, with a large user-base, is verifying that the user is running in the correct environment with the appropriate setup prior to attempting to load the PC application. For this reason, current MIS applications that utilize WordPerfect Toolkit are primarily in-house. As described in Section 4, some of this initial setup has been completed (verifying Reflection configuration, etc.). Development of the rest of the setup required to allow all users to easily take advantage of WordPerfect Toolkit is ready to begin.

3.0 Interfacing WordPerfect with HPDesk

Another application used regularly, by many of the SPMS/ORPS users, is HPDesk. Unfortunately, the "editor" is very difficult for many users to use. To resolve this problem, installation of PerfectDesk by MiniSoft on user's PC's is done on an "as requested" basis. PerfectDesk provides an HPDesk user with the following capabilities:

- Create/Edit messages with WordPerfect (CREATE/EDIT)
- Create/Edit WordPerfect Documents (CREATE/EDIT)
- Send WordPerfect Documents (WPSEND)
- Read/View WordPerfect Documents (READ/WPVIEW)
- Print WordPerfect Documents (PRINT)
- Convert WordPerfect Documents to text and vice versa (CONVERT)
- Copy WordPerfect documents to/from PC and E-Mail (WPCOPY)

The following example shows how a message can be created in PerfectDesk using WordPerfect and then sent to another HPDesk user.

```
XL:mail

HPDESKMANAGER (B.03.R0)  COPYRIGHT (c) Hewlett-Packard Company
1982,1989

Patricia I LOO signed on.
Sign on recorded at 2022 on 06/03/92.
Last signed off 2000 on 06/02/92.
        You have no appointments or ToDo items today

IN TRAY of Patricia I LOO on 06/03/92 at 2022
4 messages.

You have no NEW messages in your IN TRAY.

Intray > send

Subject: This is an example of using PerfectDesk by MiniSoft, Inc.
 TO: Gail Bird
 TO: Gail E BIRD / SSDC/00
    //
```

Loads WordPerfect and a blank screen is displayed

(Note: The following is a WordPerfect Screen.)

I am now typing an HP Mail message in WordPerfect. This is absolutely wonderful for editing. Now I have the power of WordPerfect for editing my messages. When I finish this message, I will press ALT-A to save this message. ALT-A is a WordPerfect macro, written by MicroSoft, Inc., that saves the message as an ASCII file with the appropriate filename. It then exits WordPerfect.

To save the message, press ALT-A

(Transfers file to the HP3000 and returns the user to HPDesk)

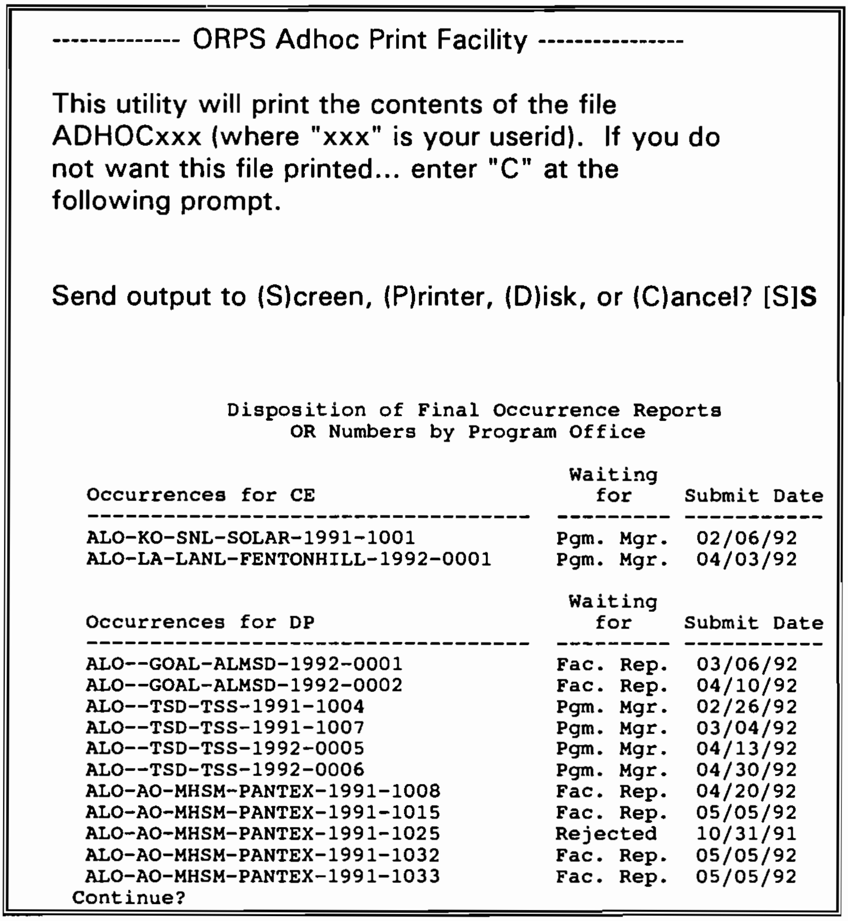
```
The message is now ready to be MAILed.

MESSAGE > list

Message.                              Dated: 06/03/92 at 2022.
Subject: This is an example of PerfectDesk by MiniSoft, Inc.
Sender: Patricia I LOO / SSDC/00                   Content 2.
This Item occupies 7 sectors of space.
Reference number: 1235931

Item  Type              Subject

  1  DISTRIBUTION LIST    DISTRIBUTION
  2  TEXT               This is an example of PerfectDesk by MiniSoft, Inc.

MESSAGE > read 2

Start of Item 2.

Subject: This is an example of PerfectDesk by MiniSoft, Inc.
Creator: Patricia I LOO / SSDC/00     Dated: 06/03/92 at 2023.

I am now typing an HP Mail message in WordPerfect.  This is absolutely
wonderful for editing.  Now I have the power of WordPerfect for editing my
messages.  When I finish this message, I will press ALT-A to save this file.
ALT-A is a WordPerfect macro, written by MicroSoft, Inc, that saves the
message as ASCII file with the appropriate filename.  It then exits WordPerfect.

End of Item 2.

MESSAGE > mail
Mailed on 06/03/92 at 2025.

INTRAY > signoff

Patricia I LOO signed off.

Thank you for using HPDESKMANAGER.


HPDESKMANAGER (B.03.R0)  COPYRIGHT (c) Hewlett-Packard Company
1982,1989

END OF PROGRAM
```

A WordPerfect document (rather than an ASCII text file) also can be transferred between HPDesk users with the WPSEND command. This command will transmit the file to the requested user as a WordPerfect document rather than as an ASCII text file. The READ command or WPVIEW command can then be used to review the WordPerfect document. This method allows the document to be transmitted with any WordPerfect enhancements that were entered in the document (i.e. underline, bold, font, etc.)

4.0 Automated Print Capability using Reflection

One of the most difficult tasks SPMS/ORPS users face is printing a hardcopy of a report. To create a more "user-friendly" print interface, the MIS programming staff developed a program that "automatically" sends a report output to the user-specified destination. This process is setup as a subroutine that can be accessed from any reporting program (MIS placed the object code in an XL library which could then be linked into any reporting application). The following shows an example of the different options for this process.

The first option allows a user to direct report output to their screen. The paging capability must be provided by each individual report.



```
------------- ORPS Adhoc Print Facility ---------------

This utility will print the contents of the file
ADHOCxxx (where "xxx" is your userid).  If you do
not want this file printed... enter "C" at the
following prompt.


Send output to (S)creen, (P)rinter, (D)isk, or (C)ancel? [S]S


              Disposition of Final Occurrence Reports
                  OR Numbers by Program Office

                                          Waiting
   Occurrences for CE                       for     Submit Date
   -----------------------------------  ---------  -----------
   ALO-KO-SNL-SOLAR-1991-1001           Pgm. Mgr.   02/06/92
   ALO-LA-LANL-FENTONHILL-1992-0001     Pgm. Mgr.   04/03/92

                                          Waiting
   Occurrences for DP                       for     Submit Date
   -----------------------------------  ---------  -----------
   ALO--GOAL-ALMSD-1992-0001            Fac. Rep.   03/06/92
   ALO--GOAL-ALMSD-1992-0002            Fac. Rep.   04/10/92
   ALO--TSD-TSS-1991-1004               Pgm. Mgr.   02/26/92
   ALO--TSD-TSS-1991-1007               Pgm. Mgr.   03/04/92
   ALO--TSD-TSS-1992-0005               Pgm. Mgr.   04/13/92
   ALO--TSD-TSS-1992-0006               Pgm. Mgr.   04/30/92
   ALO-AO-MHSM-PANTEX-1991-1008         Fac. Rep.   04/20/92
   ALO-AO-MHSM-PANTEX-1991-1015         Fac. Rep.   05/05/92
   ALO-AO-MHSM-PANTEX-1991-1025         Rejected    10/31/91
   ALO-AO-MHSM-PANTEX-1991-1032         Fac. Rep.   05/05/92
   ALO-AO-MHSM-PANTEX-1991-1033         Fac. Rep.   05/05/92
  Continue?
```

Figure 3. Example of printing report output to screen.

The second option allows the user to direct report output to their local printer. After answering two prompts the report is automatically directed to the local printer.

```
------------- ORPS Adhoc Print Facility ----------------

This utility will print the contents of the file
ADHOCxxx (where "xxx" is your userid). If you do
not want this file printed... enter "C" at the
following prompt.


Send output to (S)creen, (P)rinter, (D)isk, or (C)ancel? [S]P

Are you using an HP-compatible laser printer (y/N)? [N]Y

Please wait...report is printing.
```

Figure 4. Example of printing report output to slaved printer.

The third option allows a user to direct report output to a disk file. The user is prompted for a disk filename and then the report output is automatically directed to that file. Currently, a full path specification is required at the PC filename prompt.

```
------------- ORPS Adhoc Print Facility ---------------

This utility will print the contents of the file
ADHOCxxx (where "xxx" is your userid).  If you do
not want this file printed... enter "C" at the
following prompt.


Send output to (S)creen, (P)rinter, (D)isk, or (C)ancel? [S]D

Enter PC filename: c:\filename

Please wait...report is being printed to c:\filename
```

Figure 5. Example of printing report output to a disk file.

This automated print process has proven to be very successful. However, to incorporate this application into your production environment, some initial checks must be performed. This process requires Reflection Version 3.5 or higher. Therefore, when users logon our system, a command file is automatically executed that sets a system variable to identify the software emulation package. If the software emulation package is Reflection we then check the Reflection version, the terminal emulation, and the printer status (we can actually tell if the user's printer is offline!) Both procedures, necessary for the automated print process, have been submitted to the swaptape.

5.0 Summary

During the past few years, the need for a more user-friendly interface on SPMS and ORPS has become increasingly critical. Using inovative ideas such as what was previously described, MIS has come closer to closing the gap between the HP3000 and the personal computer...they are finally coming together at last!

Paper # - 3358

Work Flow Automation via NewWave

Leonard Block

The Apex Group
7151 Columbia Gateway Drive
Columbia, MD. 21046

410-290-1606

Introduction

With the tremendous advances in PC technology, networking, electronic mail, spreadsheets and graphics, the office environment is truly becoming an electronic arena. Companies are continuing to search for ways to maximize worker productivity by using the power of the PC workstation. This has led to the increased interest in work flow automation. If individual daily and repetitive processes can be automated, why can't processes among several people be automated.

This paper will look at the concept of work flow automation and Hewlett-Packard's HP WorkRouter work flow automation product.

What is a Work Flow?

A work flow, in general, is a group of tasks, which will be completed by a number of people in a specified sequence. Some examples of work flows:

- Proposal Generation
- Expense Report Reimbursement
- Approval of Loans
- Purchase Order Process
- Employment Applications
- Patient Admissions

Why Automate manual tasks and what benefits are companies hoping to realize:

- Reduce time-to-completion.
- Reduce errors.
- Reduce number of steps you need to perform.
- Eliminate redundancies.

HP WorkRouter Product Overview

Hewlett-Packard has developed a product called HP WorkRouter which allows the user to automate the entire manual process to improve operational efficiency. HP WorkRouter combines the latest in client/server technology, Agent Automation and GUI's to produce work flows that are flexible, dynamic and sophisticated without programming. The product is based on HP's NewWave Object oriented technology and utilizes NewWave Mail as the major vehicle for routing objects from desktop to desktop.

HP WorkRouter integrates and forms relationships between Work Objects (forms, spreadsheets, etc.), the tasks that are going to be performed on these objects, and the people who are going to be performing the tasks; the end result is a defined Workfolder Master.

Environments

Two different HP WorkRouter products are available which correspond to the two different NewWave Mail environments that they support.

<u>HP WorkRouter for DeskManager</u> is available for users who are using HP DeskManager as their NewWave Mail client

<u>HP WorkRouter for OpenMail</u> is available for users who are using HP OpenMail as their NewWave Mail client

The following represent a summary of the major features of the HP WorkRouter product:

<u>Ease of Use</u>

HP WorkRouter provides a design methodology and modular approach to help you plan your work flows and break them down into logical processing tasks.

The User Interface

HP WorkRouter looks and acts the same way as other NewWave applications. Any objects that exist on your desktop may be used with HP WorkRouter. HP WorkRouter can work with virtually any DOS, Windows, or NewWave application including electronic forms packages, word processing, spreadsheets or databases.

Dynamic Routing

HP WorkRouter decides the order of processing for its tasks based on a definition you provide. The work flow is routed from workstation to workstation through NewWave Mail, based on this definition.

Assigning Personnel

HP WorkRouter provides several choices in determining who will be performing each specific task within a work flow.

Security

To ensure integrity of work flows, HP WorkRouter allows for password protection as well as controlling what objects users can read and write during each step of a work flow.

Target Users

HP WorkRouter contains features for two different classes of users; Workfolder Designers and End Users.

Workfolder Designers

These people are primarily concerned with:

- Defining automated work flows.
- Creating Workfolder Masters.
- Testing or distributing work flows.
- Troubleshooting Workfolder Cases.

Workfolder End Users

These people are the people who will be executing specific instances (cases) of defined Workfolder Masters.

Work Flows: Logical versus Physical

HP WorkRouter is flexible enough to handle almost any type of work flow definition. Careful planning will help ensure that a work flow design will be compatible with the NewWave desktop environment. HP WorkRouter requires certain information to be supplied during the definition phase.

The following steps normally would be undertaken to logically layout a work flow. Once a logical design is in place, HP WorkRouter can be used to build the physical objects with the information that has been identified during the logical design phase.

Selecting a Work Flow Process

A work flow master contains the means of fulfilling a business process. A business process can be as simple as transporting documents throughout an office, or it can be as complex as processing purchase order requisitions for a large manufacturer.

Task Breakdown Within a Work Flow Process

Once a business process has been selected, the sequence of events to take place should be broken down into the different logical tasks.

For each logical work task that has been identified, the following needs to be considered:

- Description of the task
- Assigning Personnel

  Several different alternatives exist for assigning personnel to tasks.

  1) Identifying a specific single individual as having the responsibility for performing a specific task.

  2) Having the work flow prompt at the time a work flow is to be routed who should be the next person to receive the work flow.

3) Having the work flow use a person who has been predesignated as having a special role for that work flow such as the designer of the work flow, the initiator of a case or the person who is performing the current task in progress.

- Determining Task Inputs & Outputs

  After the work flow has been broken down into tasks, the inputs and outputs for these tasks need to be identified. The following are some of the questions that need to be applied against each task:

  1) What forms (if any) will users need to complete ?

  2) Do these forms exist in electronic format?

  3) What other software applications (spreadsheets etc.) will be required in order to process the data?

  4) What objects need to be passed from task to task?

- Setting Up Routing Flows

  Determining the order of processing among tasks and processing dependencies.

- Instructions

  After reducing a work flow into tasks, instructions can be added to help users perform their work.

Physical Work Flow Components

HP WorkRouter uses NewWave objects to build work flows. The following are the major components of any work flow:

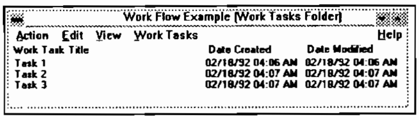
Workfolder

- Is a special type of NewWave folder object. You can file it in folders, storage objects, or in the File Drawer.
- Stores all Work Objects such as forms, spreadsheets, graphics, and other objects necessary to perform the work flow.
- Stores all specific Work Tasks that have been designed to accomplish individual steps within a given work flow.
- Controls user accessibility to objects within the work flow.
- Dynamically determines and routes the work flow to the next person who will perform a scheduled Work Task.

Work Objects Folder

- Stores all the data objects that will be used for the specific work flow.
  Any objects that exist on your desktop may be inserted in the Work Objects Folder. This is a regular NewWave folder Object which always remains inside the Workfolder.

Work Tasks Folder



Work Task Title	Date Created	Date Modified
Task 1	02/18/92 04:06 AM	02/18/92 04:06 AM
Task 2	02/18/92 04:07 AM	02/18/92 04:07 AM
Task 3	02/18/92 04:07 AM	02/18/92 04:07 AM

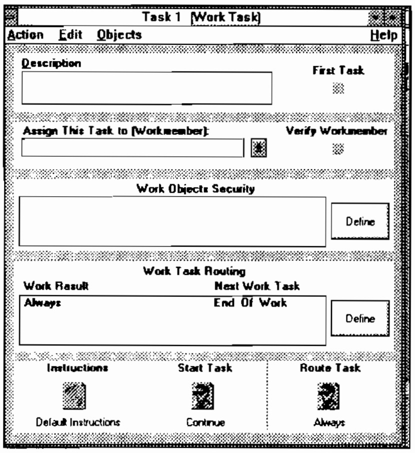
Work Tasks Folder

- Serves as a convenient container for all the Work Tasks for a Workfolder.

Work Task

The Work Task:

- Contains all the necessary information to define each specific task within a work flow.
- A section to enter the task's description.
- A box indicating whether this is the first task of the work flow.
- A section to define and verify the Workmember.
- A section to define the security of the task's objects.
- A section to define the routing that this task will use.
- An area to store the task's Instructions object.
- An area to store the task's Start Work Agent task.
- An area to store the task's Route Agent task.



Description

First Task

Assign This Task to [Workmember]:

Verify Workmember

Work Objects Security

Work Task Routing

Work Result	Next Work Task
Always	End Of Work

Instructions	Start Task	Route Task
Default Instructions	Continue	Always

Open Work Task

Enhanced Work Flows

Certain work flow requirements call for a level of processing above those provided by Basic work flows. This can be accomplished by using the Agent Task Language in the following areas:

- Setting up and performing many desktop processing options on behalf of the user. Start Agent Tasks are used for this purpose.
- Dynamically routing the work flow to different destinations based on results of actions. Route Agent Tasks can be used to return predetermined values back to the Work Task.
- Dynamically determining the next Workmember to perform a specific Work Task based on a result produced by an Agent Task Object.

Start Tasks

These are NewWave Agent Task Objects. They will begin to execute when the user Starts Work. Because it is a NewWave Agent Task, the Start Task can perform a broad range of functions, from displaying messages to performing complex processing for the user.

Route Tasks

These are NewWave Agent Task Objects. They will begin to execute when the user routes a Workfolder. Because it is a NewWave Agent Task, the Route Task can perform a broad range of functions, from displaying messages to performing complex processing for the user. The main purposes of Route Tasks are:

- To determine if the work flow is ready to progress to the next task.
- To send a Work Result back to the Work Task so HP WorkRouter can determine which Work Task the Workfolder should select as its next task.

Enhanced Routing Tables

Enhanced routing allows you to use specific Work Result values for dynamic and conditional routing instead of always having one task route to the same task.

Having the Route Task Determine the Workmember

This feature allows the Agent Route Task to determine which Workmember will receive the next scheduled Work Task.

Showing Instructions from Agent Tasks

Start and Route Agent Tasks can also be used to display the Instructions Object contained within the Work Task. This is done via a special Agent Procedure.

Other Features:

- Work Flows need to be validated to ensure they are properly designed
- Work Flow designs can be printed for documentation or design review
- Testing of an entire Work Flow design may be simulated on one desktop
- All actions performed in the definition and execution of a Workfolder are recorded in a log. The Log can be a very helpful diagnostic tool for tracing the progression of a work flow.

Sample Applications

The following are work flow scenarios that have been built with HP WorkRouter. HP WorkRouter is an object oriented product that needs to be worked with to be appreciated.

Travel Advance Request Scenario:

A user fills out the a request form, the WorkRouter Object is routed to their immediate Supervisor if the amount requested is under $1,000, or to their Department head if the amount is $1,000 or greater. When the manager receives the WorkRouter Object, they have the option of approving (authorizing) or rejecting the request. If approved, the object is routed to the Accounting Department. If rejected, the object is routed back to the original sender (initiator)

The Travel Advance Request WorkRouter demonstrates the use of several features, including:

- Hidden Objects
- Authorization
- Use of Start and Route Tasks
- Conditional Routing

Routing Example:

This example allows the user to set up an electronic "routing slip", which can have up to eight people on it. The user places objects in the routing folder when prompted to do so, fills in the routing form and adds any comments needed. Then the WorkRouter object is routed to the next person on the list. This person can make any comments on the item(s) being routed, or change them as necessary. The process continues until all persons on the list have seen the included items.At that point, the process is complete, unless the originator chooses to have the package routed back at the end. If so, the package makes one final trip.

This example demonstrates:

- Start and Route tasks
- Extensive use of NewWave Interface object
- Examples of error checking in an Agent task

County Agenda WorkRouter Example

When someone wants to submit a new item to the agenda for the next Board of Supervisors meeting, they must complete a request form. The form is then routed to the person's supervisor and to various department heads, depending on the choices made by the submitter. The supervisor and department heads can review the request, comment on it, and accept or reject it. If the request passes the scrutiny of everyone involved, it gets entered on the agenda for the next meeting. The agenda is kept in the folder marked Agenda Item Pool.

This work flow demonstrates:

Start and Route task usage
Conditional Routing
Authorization

Standard Case - Reminders on Agent Calendar

Description

This Case allows a person (Case Initiator) to dynamically schedule a reminder message onto the Agent calendar of another person. The Case Initiator customizes the routing list (Interface Object) by entering in from 1 to 8 people. The Case is automatically routed to all people with HP WorkRouter determining who is next on the routing .

The Initiator composes the reminder message and determines the date and time that the message should be displayed. The Initiator can also repetitively schedule the message weekly, monthly or periodically. A customized Agent Task is created and then is placed onto the calendar of all people on the routing list. The Work Flow determines the next person that the reminder should be routed to. When all people on the list have had the reminder message scheduled, the Work Flow folder is automatically discarded. The Work Flow also contains checks to ensure that the person's system date and time on their desktop is correct so the reminder can be properly scheduled.

Summary

Work Flow automation is just beginning to gain momentum. Many companies are entering the playing field hoping to tap the huge potential work flow market. Hewlett-Packard has developed a product HP WorkRouter that allows users to easily define work flows without the need for sophisticated programming.

Many non-technical personnel will be designing and executing HP WorkRouter work flows. HP WorkRouter provides a step by step method for planning and building work flows with adequate checks and balances to ensure designers stay on their intended course.

HP WorkRouter is not just another NewWave application, it is a fully integrated NewWave "System" that uses all the power and flexibility of NewWave's object oriented desktop. Being a brand new product, a verdict on its usefulness will not be known for some time. However, the initial reaction to the product is very positive and if things go well, Hewlett-Packard's HP WorkRouter could be recognized as one of the industry standards in work flow automation.

Using Windows and Networks to Increase Productivity and Customer Service

Kevin Wilhelmsen

Space Coast Credit Union
20 S. Wickham Road
Melbourne, FL 32904

(407) 724-5730

Introduction

It hasn't been very many years ago that most financial institutions did most of their processing by hand. These institutions hired hundreds of accountants to keep track of customer accounts and balance general ledgers. As computers developed and worked their way into the financial world, they were readily welcomed as a means of increasing productivity and service to the customers. Most of these early computer systems, as well as the systems of today, are traditionally based on the mini or mainframe concept. Financial institutions are now just beginning to look at the PC as an alternative or counterpart to the mini or mainframe computer.

In the past few years, the words *windows* and *networks* have taken on new meanings. Windows used to be what you would look out of and wish you didn't have to work today, while networks were groups of people that worked for a similar cause. With the advances in the processing power of PCs, Graphical User Interfaces or GUIs have flourished. Microsoft Windows is one of these products that has become very popular. Microsoft estimates that it has over nine million copies in use, and this figure is growing at a very rapid pace. The Windows interface brings many benefits to the operation of a personal computer. Some of these benefits are:

- **A common interface.** This allows all applications that operate under Windows to have the same look and feel. Performing common tasks such as opening a file are the same in all applications. GUIs often use a desktop metaphor to describe how applications are accessed and used. This provides users an easier way to understand and use applications.

- **Lower training costs.** Windows provides users with a point and click type of interface that is very intuitive to most users. Rather than learning numerous commands to tell a program how to perform a certain task, all the user has to remember is what buttons to push and what settings to choose to get the output they want. Once users become accustomed to how Windows operates, they are generally less intimidated with new applications and will explore the different features of applications without hesitation.

- **Higher Productivity.** Since Windows is a multitasking environment, it will allow users to have more than one program running at the same time. The user can leave one program and use another with just a click of the mouse. Being able to rapidly switch between applications, the user has the ability to accomplish several things at once.

- **New applications under development.** Applications of all kinds are being developed for the Windows environment. Old DOS applications are being re-written to take advantage of the graphic interface and in the process are becoming more powerful and easier to use. New applications are appearing that could not have been created without a graphical interface. These applications often provide us with visual information that was not before possible.

The Macintosh computer by Apple was one of the first computers to become popular that incorporated a graphical user interface. The Macintosh has become very popular for working with graphics design and desktop publishing. It has also become very popular among users that are not very computer literate due to its ease of use. Other GUIs have also recently emerged such as OS/2 and Geoworks. Geoworks like Windows runs on top of DOS while OS/2 is its own operating system.

On the other side of the computer industry is another technology that has been around for some time, but is now growing in leaps and bounds. Networking of PC computers is becoming almost as common as owning a computer. Anywhere that a company has more than three or four computers together, they are generally networked. The number of networks installed grows at an astounding rate each year.

Networks bring two basic benefits to its users: sharing of resources and communication. Networks allow users to share several resources that have traditionally been accessible to only one user. Printer sharing is the most common resource that networks share. By allowing users to easily share an expensive laser printer or color plotter, makes it easier to justify spending the money on such expensive equipment. The biggest advantage of networks is the ability to share files. By placing large applications on network file servers, several users can access programs without needing to install a copy on each computer in the company. This also helps keep the costs of upgrading users' hard disks to a minimum. Other types of resources that users can share are fax boards and data bases.

The other benefit of networks is communication. Users on networks can readily communicate with other users through electronic mail, messaging, or the latest development of digitized voice. Users often have the ability to send E-mail to others all over the country. Using this communication ability, a user can be part of a work group that will allow several people to access and work on the same files and projects.

Until recently these two technologies have not typically been integrated with each other. There have been several problems that had to be solved: management, performance, impact on the network, etc. Due to the fact that these technologies are becoming so popular, it is important that they be integrated with each other so that networks can provide the communication and sharing of resources, and Windows can provide the simple but powerful interface to the complex tasks behind the scenes.

The Need for More:

Working at Space Coast Credit Union is different from working for most other financial institutions. In my opinion, most financial institutions have traditionally been very conservative in their use of computers and technology. They tend to be more cautious about the technology they use and purchase. They typically wait until the technology has been proven and tested before implementing it. Space Coast Credit Union has a different philosophy regarding the use of technology. The management and board of directors take a pro-active approach to solving problems using technology rather than re-active. This has provided many opportunities to experiment and try out new products for providing better productivity and increasing service to the members.

Most of our users require access to our on-line transaction processing software (OLTP) that resides on an HP 3000 series 960. Terminals were provided for every user in the credit union that needs access to this software. This was fine for most users since this is all that they required, but it soon became evident that we had a need for more. Users needed access to word-processing, spreadsheets, desktop publishing, charting, and graphics applications. This need was driven mainly by the departments that were more concerned with the operation of the credit union rather than the departments that worked with the membership; though all departments were starting to cry for these types of applications.

Over the years, PCs were installed in several strategic locations to provide access for many users and giving them at least the access to some PC based software to help assist them in their work. These PCs were usually shared by several people and often in great demand. Over time several PCs were purchased and a local area network was installed. This helped reduce some of the problems but not all of them. Several users that needed access to the OLTP software as well as the PC would often have both a terminal and a PC on their desk. This problem was solved by installing Reflection on the PC. This created another problem in that the OLTP software required a formatted screen and putting Reflection into this screen required another user to set up his device number so that it could be used. If the user needed to run some software that required him to exit Reflection, he would have to go through this process again when he re-entered Reflection.

Since our credit card accounts and transactions were handled by a credit card processor, there was also the need to provide a means of access to their system for administration of the credit card accounts. They provided this access to us through an administrative terminal, but we had too many users that needed access to this terminal. It became necessary to install a 3270 gateway on the Novell network. Even with access over the network, there were still many users that needed access that did not have their own PC and would have to use others.

At this point it was becoming increasingly clear that we needed to provide each user with a PC computer that would give them access to our HP 3000, 3270 Gateway, as well as all the PC applications without having to go elsewhere to find these services and without having to exit any of these applications. More than just providing these users with a PC, we wanted to give them the ability to quickly switch between applications and provide an environment that was user friendly and easy to use. In effect what we needed was more computer hardware, multitasking workstations, a friendly user interface, and more connectivity between the computers and the mainframes involved.

The Answer:

If we were to try to find this type of system several years ago, it would have been an impossibility, but we could see that it was now time to bring networks and GUIs together to provide us with what we wanted. The first step was to define what was needed and what products would be used. There were three basic areas that needed to be covered: the type of network to use, the configuration of the PC workstations, and what GUI would be used.

Network. Due to the fact that the Novell network was already in place, the most logical choice was to remain with Novell and upgrade from the 2.15 SFT version that was installed to the latest Netware 386 3.11 version. Since the network had not been managed very well, the two ThinNet segments were way over their recommended length. This did not help in the performance of the network. To help resolve this problem, we decided to convert most of the users on the ThinNet to 10BaseT. This allowed us to continue to centralize our wiring while reducing costs and cable length problems. A 16 bit network interface card was chosen as the standard to improve network performance. To help with printing problems, we chose to install HP's new network interface cards in the existing Laserjet printers. This helped overcome the bottlenecks of workstations using the remote printer program, and also helped eliminate any incompatibilities that might arise with Windows on that workstation.

Workstation. Since most GUIs are very demanding on the systems that they run on, it was important not to purchase systems that would cause the user to be frustrated by the performance. It was decided that the minimum configuration would be a 33 MHz. 386 system with cache memory, a SVGA graphics card and monitor, and 40 Meg. plus hard drive. Since the prices of 386 systems have been dropping rapidly and due to the modular design of the PC, we decided that these systems could almost be classified as disposable. What this means is rather than purchasing the more expensive name brand systems, it would be cheaper to buy a quality mail order PC and keep a few on hand for parts replacement. Since most of these systems have a 1 year warranty, we could replace the failing component with the spares and send the failed ones back for warranty replacement. After the warranty period, we were only looking at the cost of an occasional component replacement rather than the high cost of annual maintenance.

GUI. The decision as to which GUI to select was a tough one. Each one had its own advantages and disadvantages. Microsoft Windows ended up being the GUI of choice for several reasons. Due to the recent popularity of Windows, we were fairly sure that it would not become a dying standard very soon. The large software base that Windows already had was a big factor in our decision as well as the many developers just beginning to introduce new applications. Due to the fact that Windows is still running on top of DOS was a disadvantage in the stability of the product, but with the latest release of Windows 3.1 and development of Windows NT and the Win32 standard, we felt secure that the future of the product looked bright. With the recent enhancements of Windows 3.1 and its networking abilities and Microsoft's commitment to adding even more network support to future products, this resolved several of our concerns and problems.

Merging the Two Technologies

The first step in setting everything up was to get the network upgraded and configured. The Netware 2.15 file server was a Compaq 386/20 with 3 Meg. of memory. We decided to upgrade the file server to a 386/33 with 8 Meg. of memory and more hard disk space at the same time as the Netware upgrade. 10BaseT concentrators were purchased as well as a slew of 10BaseT cards for the workstations.

Since we had gotten word that our OLTP software vendor was writing an enhancement of their software that would run under Windows and communicate with the HP 3000 over the network rather than through serial cable, we decided to prepare for this now. Our decision to upgrade to Netware 3.11 was already paying off because it would route the TCP/IP packets that the HP used for its networking. After doing some research and sorting out the conflicting answers I received, I finally discovered what was needed on the HP 3000 for it to support networking. Because we had a 960 we couldn't use the Lanic card installed for the DTCs as our access for the network. There was some confusion about this since apparently the smaller systems will let you do this. We then ordered another Lanic card and HP's ThinLAN/XL Link software. This enabled us to do virtual terminal as well as process to process communication that our software vendor would be using.

There were a few other problems that had to be discussed and solved regarding the networking side of things. This was the use of both Novell IPX packets and HP's TCP/IP networking protocols over the same network. To provide access from the PCs to the HP 3000, there were a couple of different packages that were being considered. First of all was Walker Richer & Quinn's connection 3000 product and HP's Network Services 2.1 for Netware. Both of these products effectively provided the same capabilities. We chose to use the Walker Richer & Quinn's product due to support. We felt that PC products and PC software demanded a more responsive support from the company providing them, and

Walker Richer & Quinn has traditionally done this both with their terminal emulation software and with their connection 3000 software. HP on the other hand has a mainframe software support perspective, and we felt that we could not get the support we wanted from HP to resolve any problems that may arise with the networking software.

Because we have nine different branches besides the main office, and those branches were served by multiplexers and 56k digital lines, it was required that we upgrade those links to the branches with Ethernet bridges. Since the multiplexers were leased from Codex, we upgraded the lease to use Codex Etherspan bridges. We also had a T1 circuit installed from the main office to the phone company's central office, and then let the phone company break out the T1 into the 56k circuits to the branches. This became our WAN. Now that we had network connections to each branch, we could place our DTCs at the branch for the serial communication that was still needed.

Even though the 56k circuits are really too slow for Netware networking, they work fine for the small packets of data the HP 3000 creates. Unfortunately, this is not enough band width for most Novell networking. To minimize traffic over the 56k digital circuits to the branches, we elected to place a Novell file server at each branch. That way most of the heavy network traffic was localized, and we still had the ability to manage the file servers and transfer files and E-mail over the bridges without creating too much traffic for the bridges.

It was questioned as to whether or not we would need the HP 3000 DTCs at the branches when our software vendor finished the development of their Windows based software since this would provide a direct connection to the HP 3000. We elected to leave most DTCs in place because this was not a supported feature at the time of implementation. It would also probably take a while to phase out the need for serial printers and provide a printing solution at each branch over the network using the Netware file server and PC printer.

Since we had elected to use Windows as our user interface, there were several problems that had to be resolved with regard to running on a network. When we initially began installing, Windows 3.0 was the current version. The procedures for installing and running Windows on a network were very vague, not to mention the concerns of how to manage each Windows user. After reading the manual and several magazine articles on the subject, I discovered that there are essentially three different ways of running Windows on a network. The installation of Windows remains pretty much as what is outlined in the Windows manual. This explains how to copy and expand all files on the installation disks to a specific directory on the network. How each users is setup to run Windows differs.

The first option is to run the setup from the server and install Windows on the users' local hard drive. This provides the best performance when loading and running Windows, but does not allow for easy management. This also allows the user to run Windows without being logged into the file server.

The second option for setting up Windows is using the "setup /n" option. This allows the user to install only the modifiable files on his local hard disk or private directory on the network. When installed on the users private directory, it is much easier for the administrator to make changes and add new applications. This still causes problems in managing Windows because as the number of users grow, the more user files there are that may need to be modified.

The last option is to use the "/n" option of setup to install in a common directory for all users, and then mark the files "read only". This only works on networks that have all the same hardware configuration. This is the easiest to manage because all users use the same files and any change to the configuration is global. The biggest disadvantage of this setup is that the users are unable to customize the desktop to there own liking. Each time they run Windows, the groups are always the same and the colors are what was set by the supervisor.

The option we chose was a combination of the second and third options. Each user has Windows installed in their private directory and most of their group files are there also. The network applications reside in a group that the user can't modify. This is located in the shared Windows directory. When a new application is added, it needs only to be added to the network applications group. Users are free to create other groups as they wish. This was a good compromise between management and flexibility.

With the introduction of Windows 3.1, more management and flexibility for networking Windows was added. The best source of information on networking Windows is from Microsoft themselves. Their Windows Resource Kit contains large amounts of information on how Windows operates, what files it uses, and how to modify most of those files for your installation. I have also heard talk, through Compuserve, of a Windows login utility that would bring much needed management to Windows on networks. If the talk on Compuserve is correct, it should be released in the 3rd quarter of this year.

Did We Succeed?

The question of whether we succeeded in integrating Windows and networks together to provide more productivity and customer service is a hard one to directly answer. We have Windows running on the network and most of our users have PC workstations that provide them with the applications they need without going somewhere else. As for productivity, most of the users seem to be doing more. Perhaps the best way to answer this question is to tell you about one of our departments called the information center. This department started out essentially as a phone center where they would simply answer the phones and direct the members to the respective departments within the credit union to have their questions answered. Our vision of this department was for them to be able to help a member with any service without having to forward the call.

The department initially started out, as most departments, with just terminals. The access that they had was limited just to the on-line system. They had a PC in the department that they used for occasional word processing and access to the Visa administrative terminal. Recently we moved this department into a new area, and at this time decided to provide them with the new Windows workstations. For access to the OLTP software on the HP 3000, we set them up with Reflection for Windows since the network access was not ready. They also had access to Word for Windows, Excel, Pagemaker, Microsoft Project, and Windows Draw. We then gave these users a couple of days training in the use of Windows and Reflection for accessing the on-line system which was their immediate requirement.

We noticed that the users took to the Windows environment quite readily and were able to get up and work in Reflection with the OLTP software without too many problems. Once the users became familiar with Windows, we showed them some very basic features of Word for Windows. We started by showing them how to print letters and address envelopes, as well as how to cut and paste information from the Reflection screen to other applications. They seemed to take to that quite readily and began to recognize the benefits of working under Windows. We noticed that they would begin to leave several applications running and switch between them. While training the users in Windows, we encouraged them to play the solitaire game that comes with Windows. This helped the users to develop mouse skills and become more familiar with the Windows environment. This is probably one of the few times that an employer may encourage employees to play video games while on the job.

After becoming more familiar with Windows, the users began to explore the other aspects of Windows and looking into other programs. Because their jobs revolved around giving members information from the on-line system as well as mailing information to members, they seemed to stay mostly within Word for Windows and Reflection rather than venturing into Excel and other Windows applications. Because of the increased productivity of these employees, we were able to avoid hiring two new personnel. As we install and set up more users with PCs and Windows, we see the same thing happening over and over. The users accept and adapt to Windows quite readily. They seem more interested in working with it and learning how to work within the programs. Our problem now is keeping up with the request of the users wanting these Windows workstations, and giving them the training on all the different Windows software to help them be more productive.

Many of our users are still just getting the feel of Windows while others have taken it and are doing fantastic things. Each case is different and the integration will be an on-going process as new products are developed and applications get smarter.

Paper # - 3360

Utilizing Graphical User Interfaces in NewWave

Leonard Block

The Apex Group

7151 Columbia Gateway Drive Suite F

Columbia, MD. 21046

(410)-290-1606

Introduction

A picture is worth a thousand words....

This old cliche is very applicable to todays PC software development arena. Faster and more efficient CPU's , better memory managers and high quality graphic monitors are making the task of bringing pictures to one's PC a practical and cost effective solution.

Almost everyone associated with personal computers today is aware of the presence of Microsoft's Windows software. Until recently, only MS DOS could be counted on as being standard software on a workstation. Due to the enormous popularity of Windows, it too is now standard software on many workstations. Hardware vendors are now offering Windows as part of new PC purchases along with DOS. With the advent of sophisticated LAN database engines and powerful Application Program Interfaces (API's), the software applications development arena for PC's is clearly moving in the direction of Windows based applications.

If one looks at any of the major technical trade magazines, they will notice the multitude of advertisements for new Windows based applications. Even strong traditional DOS based winners such as Lotus and WordPerfect are being overhauled to take advantage of the GUI of Windows in order to capture a piece of the huge market demand for such applications.

One of the more interesting software packages struggling to find its niche in this Windows market is NewWave from Hewlett-Packard.

Why do I need NewWave if I have Windows ?

A year ago I addressed this same question in a paper highlighting the power of the Agent Scripting Language in using it to automate simple, complex and repetitive tasks in automating work on one's desktop. I believe that the paper was successful in making people more aware of NewWave as a desktop manager and applications development tool under Windows.

As Rodney Dangerfield says " I can't get no respect" and NewWave is still finding it hard to gain acceptance in the Windows market. Well here I go again trying to gain some respect for NewWave and its capabilities. This time I will be zeroing in on its Graphical User Interface for forms design and data capture.

Beyond the iconics...

Most people are aware of NewWave's iconic representation of objects. HP has been hard at work enhancing NewWave to the point where users can build their own custom objects through an easy to use GUI that is new to the latest release of NewWave.(4.0) What is nice is that one does not have to possess sophisticated Windows programming skills. What is even nicer is that the objects that are developed can be utilized with the Agent language to create powerful applications at a low cost.

This paper will address three areas where the Graphical User Interfaces in NewWave can be very beneficial in producing useful applications without the requirement of being a "heavy duty" 'C' or Windows programmer.

Forms Design and Data Capture

As is still the case, many applications are only as good as their interface to the user . Many well programmed technical routines go unappreciated due to poor screen design and data input programming. With the power of GUI's, the toolbox to present user friendly and high quality screens is greater than ever.

NewWave contains an object called the **NewWave Interface Object** that allows you to build sophisticated data entry screens and dialog boxes very easily that can be used with Agent tasks. The NewWave Interface object provides an alternative to conversational windows available in Agent tasks. The advantage of Interface objects is that you do not have to use Agent task language statements to draw a sophisticated form. A conversational window requires you to write Agent task language statements to create each window for a task.

The NewWave Interface Object offers a large variety of dialog boxes and fields that can be constructed to produce high quality electronic data input forms.

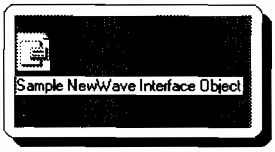
Functions of the NewWave Interface Object

- Allows a user to select from several choices.
- Prompts for information an Agent task needs to use.
- An Agent task can open an Interface object to display.
- An Interface object can perform an Agent task.
- The Agent can set values in predefined fields, buttons and boxes.
- The Agent can retrieve inputted values from fields, buttons and boxes.

How to Begin

To begin the creation process, one just creates an instance of a NewWave Interface Object via the standard creation process for any NewWave Object. The following icon will be displayed for the NewWave Interface Object:

Illustration 1 - The NewWave Interface Object



Modes of Operation

NewWave Interface has two modes of operation -- edit mode and work mode.

In the **edit mode**, you create and edit the dialog box.

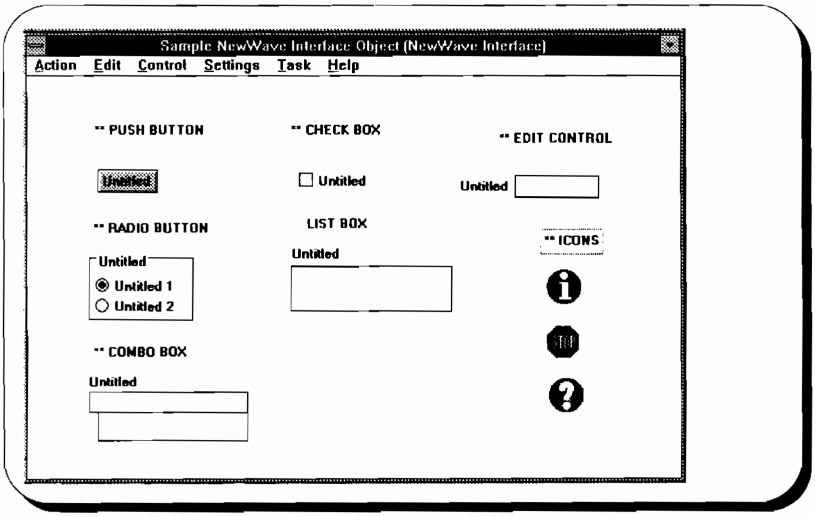
In **work mode**, the user enters information in the dialog box you created and no changes to the design are allowed. You can make the edit mode inaccessible to the user. This will be explained later on.

The mode for the form is controlled from the Settings Menu of the Interface Object. Toggling the selection **Edit Form** lets the user switch from **edit mode** to **work mode.**

Sophisticated Dialog Boxes at the click of a mouse...

Creating dialog boxes are easily created by selecting a type of box from the Edit menu and clicking in the part of the object where you want it placed. The following explores each of the dialog boxes in more detail.

Illustration 2 - Dialog Boxes in their initial states before customization.



Description and Functions of Dialog Boxes

Push Button

- **Function** - Custom buttons can be produced that when pushed instructs an Agent Task to proceed with processing.
- **Label** - The user can label the button.
- **Location** - The user can position the button through the setting of X and Y axis coordinates.
- **Multiple Buttons** - Multiple buttons can be placed on a screen with one button identified as a default button should return be pressed.

Edit Control

- **Function** - Used primarily for user data input.
- **Label** - The user can label the field and controls the place for the label. Labels may be placed to the left of the field or above the field.
- **Size** - The vertical and horizontal lengths can be set in the design of the field. The user even has the option of scrolling information in a field should the field require an extraordinary amount of data.
- **Location** - The user can position the button through the setting of X and Y axis coordinates.

Radio Button

- **Function** - Standard Radio Button selection dialog box.
- **Label** - The user can add up to 8 button label selections. The number of buttons per row is also set by the user. A group label for the entire radio button cluster is also created.
- **Location** - The user can position the button through the setting of X and Y axis coordinates.

Check Box

- **Function** - When a checkbox is clicked, it becomes marked with an 'X'.
- **Label** - The user can label the field .
- **Location** - The user can position the button through the setting of X and Y axis coordinates.

List Box

- **Function** - Values can be selected from a predefined list set up by the designer.
- **Label** - The user can label the field and controls the place for the label.
- **Size** - The vertical and horizontal lengths can be set in the design of the field.
- **Selection** - The box can be setup so a single value or multiple values may be highlighted with the mouse.
- **Location** - The user can position the button through the setting of X and Y axis coordinates.

Combo Box

- **Function** - Values can be selected from a predefined list set up by the designer. This is similar to the List Box Function.
- **Label** - The user can label the f ield and controls the place for the label.
- **Size** - The vertical and horizontal lengths can be set in the design of the field.
- **Selection** - The box can be setup as a simple or drop down combo box.
- **Location** - The user can position the button through the setting of X and Y axis coordinates.

Text

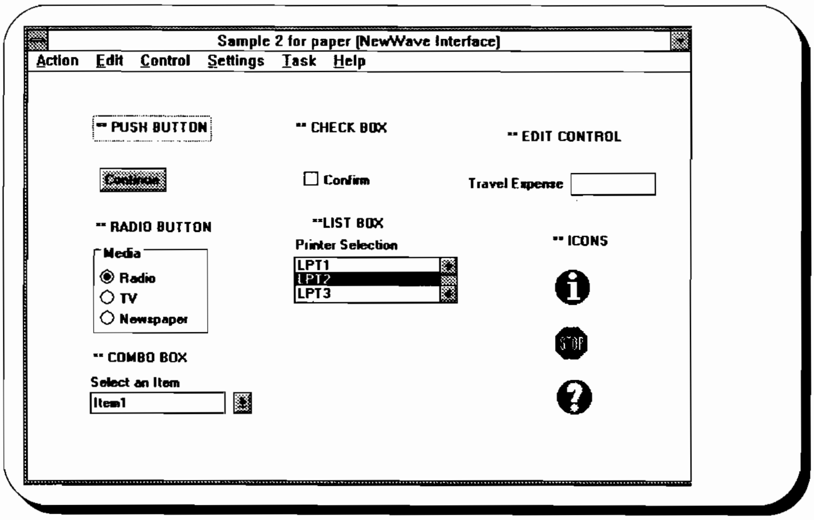
- **Function** - Any free form text may be placed on the screen. The title : ** PUSH BUTTON, ** CHECK BOX etc. are all done with the text function.
- **Size** - The vertical and horizontal lengths can be set in the design of the field.

Icons

- **Function** - 3 different icons can be placed in the object for emphasis. A question mark, stop sign or information sign may be displayed in an object. Multiple icons are allowed.

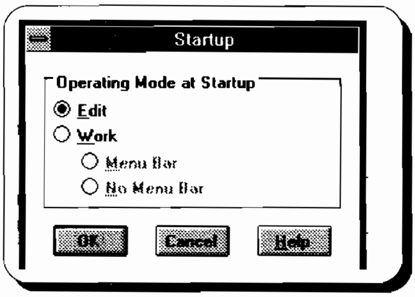
The following illustrates each of the dialog boxes shown in illustration 2 after they have been customized by a user.

Illustration - 3 Interface Object after Customization



Startup

Illustration 4 - Startup selection for the Interface object



Operating Mode at Startup: [✓ Edit] Work (Menu Bar, No Menu Bar)

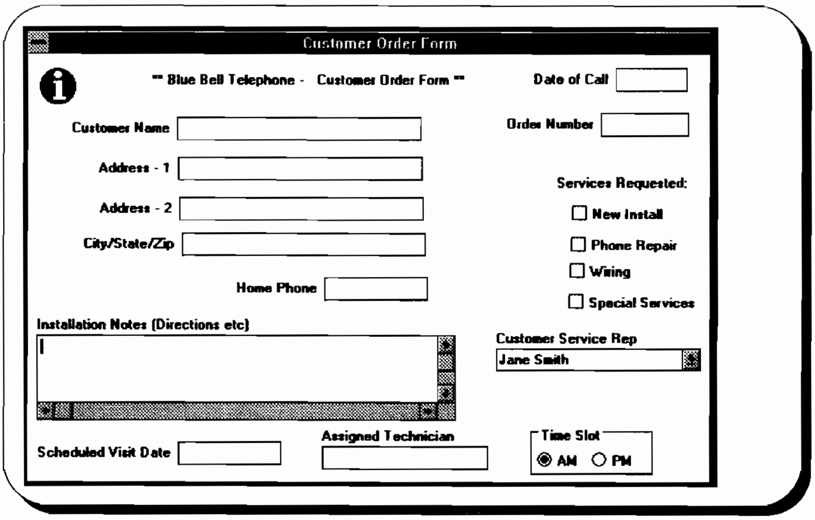
The Interface Object can be set up so it is automatically placed in work mode with no Menu bar showing. This will stop users from editing the dialog boxes in a production environment.

Sample Applications

So as one can see, many different options exist for creating custom data input screens. The following represent some examples of how all of these different dialog boxes were combined to produce electronic data entry forms.

Case 1- A data entry form to be filled out.

Illustration 5 - Customer Order Form to record telephone service orders.



Customer Order Form

** Blue Bell Telephone - Customer Order Form **

Date of Call	
Customer Name	
Order Number	
Address - 1	
Address - 2	
City/State/Zip	
Home Phone	
Installation Notes (Directions etc)	
Customer Service Rep	Jane Smith
Scheduled Visit Date	
Assigned Technician	

Services Requested:

☐ New Install

☐ Phone Repair

☐ Wiring

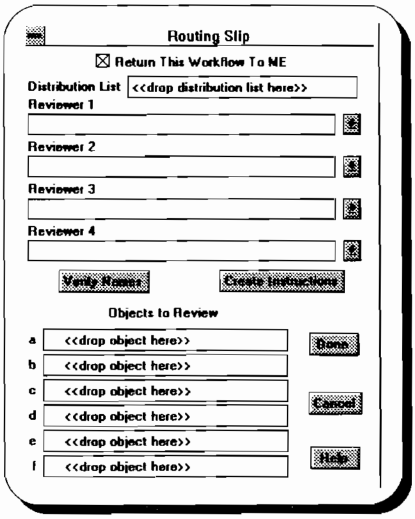
☐ Special Services

Time Slot: [✓ AM] PM

Case 2- A routing slip for people to review material.

Illustration 6 - Customized Routing Slip

This case also demonstrates additional capabilities of the NewWave Interface Object. Certain NewWave objects such as distribution lists and textnotes may be dragged into the NewWave Interface object. Values from these objects can be used as data input for the NewWave Interface Object.



Routing Slip

☒ Return This Workflow To ME

Distribution List <<drop distribution list here>>

Reviewer 1

Reviewer 2

Reviewer 3

Reviewer 4

Objects to Review

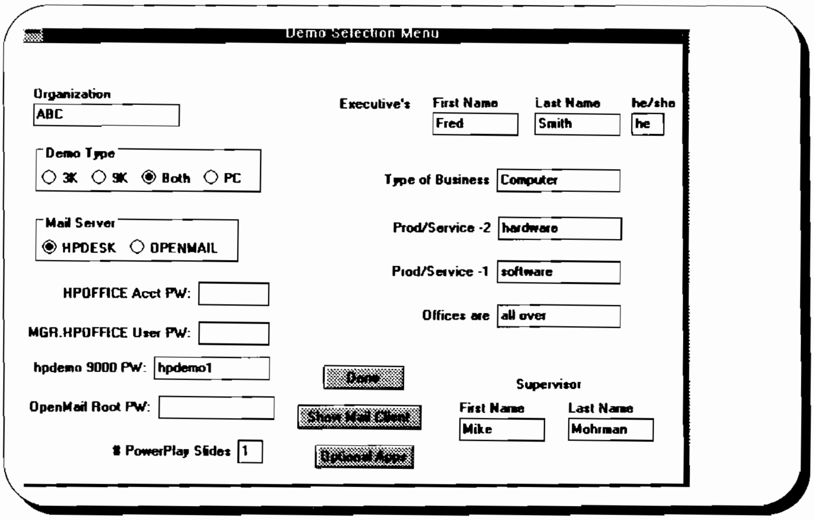
a <<drop object here>>
b <<drop object here>>
c <<drop object here>>
d <<drop object here>>
e <<drop object here>>
f <<drop object here>>

NewWave Interface and the Agent

What really makes the Interface object so powerful is that your fully designed form can interact with the NewWave Agent. The Agent can open up the Interface Object, Set Values in fields, retrieve values in fields, call DLL's, wait for buttons to be pushed and then react accordingly. The following is some excerpts of an Agent Script that utilizes the NewWave Interface Object. Pay close attention to the lines highlighted in bold. They demonstrate the interaction of the Agent with a defined Interface object. A picture of the NewWave Interface Object that was used with this script follows the script.

```
Focus Folder "The Demo Factory"
Select Newwave_Interface "Demo Selection Menu"
Open
Focus Newwave_Interface "Demo Selection Menu"
Prompt "Please Fill Out Your Selections, Then Press 'Done'"
While Continue#= "N"
Focus Newwave_Interface "Demo Selection Menu"
Wait_For_Pushbutton
Item#=Getpushbutton()
If Item#="   Done   "
 Co#=Gettext("Organization")
 Mail_Server#=Getbutton("Mail Server")
 Tob#=Gettext("Type Of Business")
  Wo#=Gettext("Offices Are")
  Info1#=Gettext("Prod/Service -1")
 Info2#=Gettext("Prod/Service -2")
 Ln#=Gettext("Last Name")
 Lnb#=Gettext("Last Name ")
 Fn#=Gettext("First Name")
 Fnb#=Gettext("First Name ")
 Mail_Server#=Getbutton("Mail Server")
  Set_Text "Demotype" Demo_Type#
 Set_Text "Demofor" Co#
 Set_Text "Mserver" Mail_Server#
 Set_Text "Slides" Numsl#
endif
If Item#="Optional Apps"
 Focus Folder "The Demo Factory"
 Select Newwave_Interface "Optional Apps Menu"
 Open
```

Illustration 7 - NewWave Interface and the Agent



Organization: ABC

Demo Type: 3K 9K [✓ Both] PC

Mail Server: [✓ HPDESK] OPENMAIL

HPOFFICE Acct PW:

MGR.HPOFFICE User PW:

hpdemo 9000 PW: hpdemo1

OpenMail Root PW:

PowerPlay Slides: 1

	First Name	Last Name	he/she
Executive's	Fred	Smith	he

Type of Business: Computer

Prod/Service -2: hardware

Prod/Service -1: software

Offices are: all over

	First Name	Last Name
Supervisor	Mike	Mohrman

Other NewWave Graphical User Interface Considerations

Presentations

NewWave can be used as an effective tool in creating visual presentations. Remember, any Windows application can be run from NewWave. With the assistance of two additional Windows packages, NewWave object windows can be captured and edited. These edited screen captures can then be printed and packaged to form effective graphical presentations. The following procedure can save a user countless hours in having to reproduce screen layouts etc. in a special graphics package.

Step 1 - Open Up the NewWave Object window to capture

Step 2 - Run the Windows Program PaintShop. Paintshop is a ShareWare application. If you can obtain a copy of this program it will be well worth the effort in creating presentations. Paintshop allows you to capture screen images and saves the output to a variety of different graphic formats. Among them are : bmp, pcx, wpg, tiff and others.

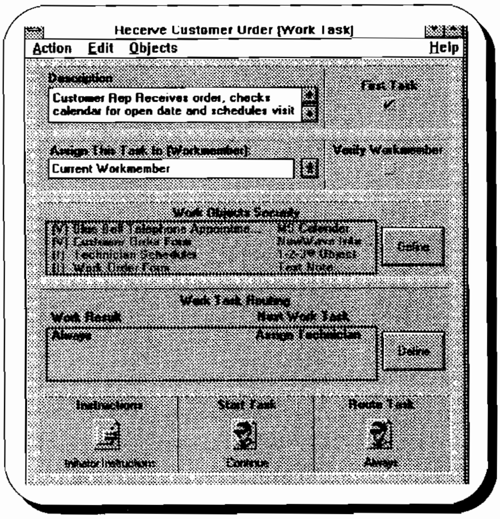
Step 3 - Open up Paintbrush (Standard with Windows) and import the graphic you just captured with PaintShop. Paintbrush is an excellent tool for touching up and refining your graphic image to meet your presentation output needs (Laser Printer, Plotter, etc.). The end result is a fully edited visual graphic of your NewWave object to be used in an office, or boardroom presentation.

Unix Motiff for NewWave

Often one hears about the look and appearance of screens built with Motiff for Unix based platforms and why can't Windows screens look that way. Motiff screens possess a clear, 3-D, chiseled steel look. They are crisp and clear and far superior in their look to screens built with standard Windows development packages.

In a recent development effort, this Unix Motiff look was achieved through the use of a package from Borland Inc. The package is called, **Resource Workshop.** Resource Workshop lets you design Windows dialog boxes through graphics oriented visual resource editors which make it easy to design and modify resources. The nice thing is that this package is inexpensive, under $100.

Since NewWave and Windows use the same visual base for creating dialog boxes, you can now build NewWave dialog boxes with Resource Workshop to produce screens that look like the following:



Description
Customer Rep Receives order, checks calendar for open date and schedules visit

Assign This Task to [Workmember]:
Current Workmember

Work Objects Security

```
[V] Ohio Bell Telephone Appointme...   HP Calendar
[V] Customer Order Form                NewWave Inbo...
[ ]  Technician Schedules              1-2-3® Object
[ ] Work Order Form                    Text Note
```

Work Task Routing

Work Result	Next Work Task
Always	Assign Technician

Summary

Last year I advocated that the small investment in NewWave ($125.00) was well worth it based on the powerful Agent Scripting Language alone. With NewWave 4.0, it is even a better deal. The introduction of the NewWave Interface object as a vehicle to create easy to use dialog boxes without any programming knowledge is one of the best kept secrets. The fact that these dialog boxes may be manipulated with the NewWave Agent make developing Graphical front end applications in NewWave a realistic possibility that should be explored by any Windows user.

Maybe after some research and testing you too will give NewWave just a little respect....

Paper No. 3361

Personal Computer Management, Audit and Control

by

Dirk Huizenga

Dynamic Information Systems Corporation
5733 Central Avenue
Boulder, CO 80301
303-444-4000

Current View of Personal Computing

The personal computer is an extension and improvement of past computer hardware and software. In the framework of a business, however, data processing departments view the personal computer (PC) as a threat to the traditional control of data and its processing. This view is not totally incorrect, but realize that microcomputers are here to stay and have their place in the Data Processing (DP) and Information Systems (IS) areas of a company.

The normal response of IS departments has been a sincere effort to support users with personal computers and to implement strategies and policies to control microcomputers. This has proven impossible using normal DP practices partly because of the lack of understanding of what personal computing is, what programming on one is (e.g., building a spreadsheet model), and what PC users are trying to accomplish. In the past, one of the major controls of DP was the relative ignorance of users. With the success of computer literacy this is no longer a means of control, especially as a little knowledge can be a dangerous thing. This has created a need to control and audit certain areas of microcomputer use. To do this without destroying the individual autonomy and entrepreneurship that has been a boon to personal computing is a very difficult, but necessary task.

Need for a New View of Personal Computing

The critical issue hinges on recognizing that the microcomputer, and personal computing, is only the beginning of a technology that extends the boundary of computer applications every day. We have yet to see the hardware and the software that will affect computing in the next ten years. This uncertainty and unpredictability make planning for microcomputers impossible, yet essential.

To address this impossible task, the IS department needs a conceptual view of personal computing. The essential elements of this view are:

- that it must be applications oriented, not technologically oriented;

- that it be based on an understanding of the microcomputer and its appropriate uses both now and in the future;

- and above all, that it positions the microcomputer in the context of all computing.

In addition, we need to think of applications that go beyond individual uses of microcomputers, and consider end users as an extension of the IS department.

Users of Personal Computers

Using Personal Computers

Probably the most wide spread use of the microcomputer is in developing spreadsheets. In fact, it was the spreadsheet program that caused the microcomputer explosion among nontechnical people. As users become more experienced, they tend to customize these programs using macros and other advanced features. Other uses include database programs, and traditional programming languages to accomplish unique tasks.

Two areas that have grown significantly are graphics and data communications with other computers to download data to microcomputers for analysis. These areas are likely to continue to grow in the future.

Defining User Levels

In the past, users were segregated by their level of understanding of PC technology. This was necessary when microcomputers required champions to introduce them into organizations. A new way of segregating personal computer users is necessary now that microcomputers have gained wide acceptance. One method may be to put users and their information[1] into a benefit and beneficiary matrix (Table 1). The beneficiaries are divided into organizations, that are made up of departments, which in turn consist of individuals.

Table 1. Benefit vs. Beneficiary Matrix.[2]

	Beneficiary		
Benefit	Individual	Department	Organization
Efficiency	I	II	III
Effectiveness	I	II	III
Transformation	III	III	III

The roman numerals (I, II, and III) relate to the spheres of influence (domains) that information affects. **Domain I** information is personal in nature and includes memos and letters in a word processor, small spreadsheets, and simple data files or tables. **Domain II** information includes departmental databases, large spreadsheets, graphics and simple publications. **Domain III** knowledge consists of large company-wide databases and the programs to transform and manipulate them, desktop publishing, sophisticated publications, and connections other information sources.

Defining Benefits

Efficiency is the ratio of output to input. PCs can allow a secretary to produce much more than can be done on a manual typewriter. An accountant can compute, in minutes, amortization tables that before took hours. Efficiency can also mean the ability to search thousands of pages of text in seconds. **Effectiveness** is using data to produce the desired impression or response. For example, presenting a map with a dot for each sale with the ability to point to more detail on each sale. Using graphics to present data is much more effective than a report full of numbers. To paraphrase, a graphic is worth a thousand datum. **Transformation** is to change the function or nature of data in entirely new ways to support decision-making. For instance, summarizing operational data to provide strategic planning information.

The different domains describe levels of software technology. In reality, it is the benefits, the scope of information's impact, and the magnitude of the risk that is the divider. **Domain I** is where most personal computing now resides. Most of its benefits are limited to the individual. Some examples of Domain I benefits are the speed and accuracy of writing a letter using a word processor, and the ability to communicate more effectively by including graphs and illustrations in documents.

For the functional unit (an accounting department, for example) to benefit, users must be lead into **Domain II**, which benefits the larger group. Network file servers used to share information (documents, databases) within a department is a good example of Domain II benefits.

Domain III has an even wider impact. It not only helps the individual be more productive, but encourages thinking in terms of business value, and how new products and services could benefit the whole organization. In the other domains, microcomputers were used to replace or enhance existing office equipment functions. Domain III incorporates completely new uses for microcomputers, such as using a database of documents to instantly produce new documents, or allowing people to access a document database using a touchtone phone and a fax machine. Client/server and distributed data are some of the concepts in Domain III that promise to provide the next leap in organizational information integration. Microcomputers will act as the bridge between old and new operational systems and will allow "reengineering" our businesses by changing the way we do business.

Developing Skills

Traditionally personal computer users have been self-taught using manuals, and trial and error. It is the error part of this training that struck terror in the hearts of DP managers. This fear probably contributed to some manager's resistance to microcomputers. Powerful microcomputers with graphical interfaces are allowing end users to develop applications by pointing and clicking, but that doesn't mean that the results are correct.

By applying the matrix in Table 1 to the development of user's skills, we can prevent the stagnation of users in Domain I and move them into Domains II and III. This allows end users to access data and produce decision-making information themselves while the IS department is improving or acquiring the tools that are needed to access the data. With these longer range goals in mind, user training should involve education in the concepts of computing. This provides an individual with the basis to accumulate skills on their own, and to progress into Domains II and III.

Control

The usual approach to controlling microcomputers has been bureaucratic and rigid. This causes users to develop their own informal support structure. Without the guidance of senior management, these renegade DP departments may not be serving the needs of the organization as a whole. The IS management needs to recognize Domain I uses and manage these toward Domains II and III if necessary.

Data security is usually a cause of concern among DP professionals, but need not be handled any differently than sensitive reports or memos. The responsibility is on the individual having access to that data.

Incorrect data analysis is another problem that needs to be addressed. User training and support should be at a high enough level that some of these problems will be eliminated. Audit user programs (i.e., spreadsheets, graphs, etc.) for accuracy before this using information. By analyzing advanced users, controls can be put into place before other users are introduced to these applications.

Support

Divide microcomputer support according to the matrix in Table 1. Provide tutoring for new users in Domain I, need assessment and technical consulting for users in Domain II and III. In addition, hardware maintenance is a valuable service that can have a tremendous effect on the benefits.

As microcomputer technology changes, the allocation of support services should shift along with it. Adjust support along two dimensions: support intensity (consulting versus implementing) and user activity (acquisition versus disposal). With technology changing rapidly, disposal of old technology cannot be neglected. Monitoring the needs of "advanced" users provides a way to predict future demand for support services.

Policies

Consider the following elements in establishing controls and related policies relative to personal computing:

Direction and Justification. There should be a clear statement of the business uses of microcomputers and acceptable criteria for judging proposed expenditures.

Data Administration. The organization should address, for all PC users, how data is provided and how its use is monitored to ensure both integrity and security.

Auditability, Documentation, and Security. Establish controls in each category while limiting the cost of control.

Standards. Hardware, software, and documentation standards should be established for a clear and agreed upon purpose, rather than for an abstract need for control.

The level of responsibility for each element of control varies according to the domain as illustrated in the following tables. Tables 2, 3, and 4 divide the responsibility among the individual user, departmental staff, senior departmental staff, the Information Systems department, and an internal or external auditor. The symbols used in the tables are listed, in order, from most to the least responsibility:

P	primary responsibility involves initiating, staffing, monitoring, and modifying a particular control element,
R&C	reviewing and concuring on an element,
S	support of policies and plans,
A	advising,
N	notifying.

Table 2. Domain I Responsibility[3]

	User	Staff	Sr. staff	IS	Auditor
Direction		P	R & C	A	
Justification	P	R & C		A	N
Data Admin.	P	R & C			R & C
Auditability	P	R & C			R & C
Document.	P	R & C			R & C
Security	P	R & C		S	R & C
Standards	A	R & C		P	R & C

Table 3. Domain II Responsibility

	User	Staff	Sr. staff	IS	Auditor
Direction		P	P	A	A
Justification		P			
Data Admin.		P		P	R & C
Auditability		P		P	R & C
Document.		P		P	R & C
Security		P		P	R & C
Standards		P		P	R & C

Table 4. Domain III Responsibility

	User	Staff	Sr. staff	IS	Auditor
Direction		A	P	A	A
Justification		A	P		
Data Admin.				P	R & C
Auditability				P	R & C
Document.				P	R & C
Security				P	R & C
Standards				P	R & C

Conclusions

Microcomputer training and equipment are a large part of a company's computing resources. As a result, they must be managed as carefully as any other asset or tool. Investments in microcomputer technology that does not fit into an overall plan should only be made when technology is relatively inexpensive, or when short term needs are paramount. Microcomputer management should be flexible, and the data should be in a usable and compatible format.

Uses for microcomputers are still evolving and cannot be rigidly controlled. Controls placed on personal computing should be relative to the risk to the company. The lower risk controls can be guidelines (and not followed). Where the risk is greater, more rigid rules are necessary. Domain I information should be considered personal property and left for the individual to manage. Domain II information requires basic documentation and control. Domain III information is corporate property and should be managed like any other valuable corporate resource.

Just as fire has been tamed by man to reach the stars, so can the personal computer be used to take us places that we have never been.

---

[1] The *Encyclopedia of Computer Science and Engineering* gives a generally accepted definition of **information** as data that is used in decision-making. **Data** are facts or are believed to be facts which result from the observation of physical phenomena.

[2] *Managing Personal Computing,* Index/Hammer Report, October, 1984.

[3] *Managing Personal Computing,* Index/Hammer Report, October, 1984.

Bibliography

Earl, Michael J., *Management Strategies for Information Technologies*, Prentice Hall, 1989.

Hammer, Michael, "Reegineering Work: Don't Automate, Obliterate," *Harvard Business Review*, July-August 1990.

Index/Hammer, *Managing Personal Computing*, Index/Hammer Report, October, 1984.

Inmon, W. H., *Data Architecture: The Information Paradigm*, QED Information Sciences, Inc., 1989.

Ralston, Anthony, et al., *Encyclopedia of Computer Science and Engineering*, 2nd Ed., Van Nostrand Reinhold Co., 1982.

PAPER NO.: 3363

TITLE: Using Dynamic Data Exchange to Integrate Host and Windows Applications

AUTHOR: Brian Hargus
Walker, Richer & Quinn
2815 Eastlake Avenue East
Seattle, WA 98102
(206) 324-0350

HANDOUTS WILL BE PROVIDED AT TIME OF SESSION.

Paper No. 3364

Client Server Technology Made Easy Using PPL

Tim Klooster
Dynamic Information Systems Corporation
5733 Central Avenue
Boulder, Colorado 80301
(303) 444-4000

Let's say that you have an HP3000 with PCs connected via a terminal emulation package. Your users are pressuring you for a graphical environment in their applications. Or maybe your users need to represent mainframe data on a PC and are tired of loading the data by hand into a PC package.

Client server applications may be the answer for you.

By now, everyone has heard about client server and how it will be the future for every data processing application. We've also been bombarded with solutions by countless vendors on how to make our applications "open systems".

With all the tools on the market today that facilitate distributed processing, the choice to migrate to a client server application is followed by an even more difficult decision of which tool to use. We also have the pressure to produce applications that will be platform independent. This brings up the dreaded "open systems" concept that has more definitions than we care to hear.

For our discussion, let's say that we have decided to try a simple client server application and we want to do the development internally. Rather than throw out all of our hardware, we will use a tool that will allow us to use much of our existing hardware and software.

Walker, Richer and Quinn offers Process to Process Linkage (PPL) as a tool to develop client server applications. The immediate benefit of this tool in our scenario is that it reproduces the HP programming environment on a PC. It will allow our application to be "open" in that we can easily port the communications layer of the application from the MPE operating system to either VAX or UNIX operating systems. Additionally, it operates on serial connections (RS-232), local area networks (LANs), and wide area networks (X.25). This allows our users to use the application from any connection.

Chances are that your data is stored in an Image database. To develop a client server application that is data storage independent, we would have to integrate an SQL database into the design. Unfortunately, Image does not directly support SQL. Since we want to use as much existing software in our shop as we can, let's stay with Image for the time being and plan a future port to SQL. Besides, what good is a system design without a little planned vaporware? PPL ships with a toolkit to perform Image calls, but currently has no SQL support in the form of toolkit calls. An ambitious programmer could create an SQL interface to PPL with a little work.

The programming environment for incorporating PPL calls requires a language that is Microsoft OBJ format compatible. The most popular examples are C and Pascal. The examples used here are written in C. C will also give our applications portability.

The popularity of Microsoft Windows Version 3 has brought about many new development tools. PPL routines can now be called from packages like Visual Basic, Objectvision, and Quick C. These packages allow menu driven creation of Windows applications. Although rarely can an application be created without a programmer writing some code, these tools greatly help the Windows learning curve. Anyone who has written a Windows program using C and the Software Development Kit (SDK) from Microsoft knows that it is not an environment for the faint of heart!

A.) Things to consider before starting.

When interchanging data between the PC and HP, there are a couple of differences to be aware of. Integers are represented differently on a PC than an HP. A 16-bit integer is called a SHORT data type on the PC and is represented by a WORD on the HP. A 32-bit integer is represented with a LONG data type on the PC and a double WORD (J2,I2,etc.) on the HP. Since the alignment differs on each platform, the integers must be converted when going from one platform to the other. The PPL Toolkit provides conversion routines for shorts (ICONV) and longs (LCONV).

The second difference is that many C functions on the pc require string data to be null terminated. Since the HP doesn't store nulls, you need to be sensitive to this when you port down your data to the pc. Normally, a string function that appends a null to the end of your string will handle this problem.

B.) Basic Client Server Model

The typical client server model normally consists of three modules.

The first module is the **Client module**. The client often employs a graphical user interface (GUI) that takes advantage of the graphical nature of the PC. This interface generally allows a friendlier, more flexible environment than a mainframe can provide. Although PPL is currently a DOS based tool, there are some packages that integrate DOS, Mac's, OS/2 and Unix Workstations together as a client. As my first example will illustrate, the client program may not employ a GUI interface, but rather function as a vehicle to simply bring down data for another process to use.

The second module is the **Application Program Interface (API) module**. This module serves as a bridge between the client and server modules. The purpose is to create an environment that allows easier access to the communications functions. In my example below, I create a LIB file containing functions that can be called from the client program. This allows the transport layer of the communications link to be made transparent to the client programmers. It is similar to the COPYLIB concept where an area is created to store code that can be shared among applications. Careful design of this module can create a highly productive programming environment with a smaller learning curve for new programmers. If the API is created in the form of a Dynamic Link Library (DLL), the functions are easily called from the Windows application tools mentioned above such as Visual Basic. My example executes all PPL calls from the API. I use the messaging approach of PPL rather than the Image toolkit calls to communicate with the server (host) program. This approach allows a customized environment where the server program performs multiple actions per PPL transaction. This is purely a performance consideration. The bulk of your transaction time is taken up by the communications between platforms. The more calls to PPL that you use in a transaction, the more your response time increases. This is why it's important to minimize the size of the buffers that you pass between the server and the client. Transport only the data that you need for your application. An important consideration of a client server application is to make the server connections and transports invisible to the user. The API can have special functions such as session initiation and termination without user intervention. The Windows version of PPL (beta at this writing) allows the use of Reflections script files to provide further functionality.

The third module is the **Server or Host module**. The function of the server module is to receive API calls and translate into actions on the host. The server normally stores the data and performs the more cpu intensive functions such as data retrieval, data summarizations, and complex numerical operations. The PPL driver on the server is PPLHOST. A version of PPLHOST that requires privilege mode (PM) is PPLHOSTP. The use of PPLHOSTP is heavily encouraged for performance considerations. This driver program will start the toolkit server program, PPLTOOL, if you use the toolkit or it will start your own server program if you use one.

The main rule to remember is to use the PC where the PC is strong and use the mainframe where the mainframe is strong.

C.) Implementation of the Client Server Model

This section will illustrate an application using PPL. The intention of using this example is only to give the reader an overview of some of the code neccessary to accomplish these tasks rather than furnish a complete working example. Some functions like error handling have been abbreviated for the sake of space. Example 1 describes the application in general terms and then shows code examples from the client, API, and server modules.

Implementation Example - Order Processing System

This example illustrates how to use PPL in a DOS environment. The data is stored in an IMAGE database on the HP. The pc interface is a simple C program that uses the circuit control and messaging services from PPL to exchange data with the server. Since the requirements for the host processing are fairly elaborate, I've chosen to write my own server program instead of using the Image calls built into the PPL toolkit. The advantage of this approach is a customized environment on the server that is maximized to yield optimal performance. Instead of using PPL to process each database intrinsic call that I need, I build the intelligence into the server program and use PPL to initiate the action and pass the necessary parameters to the server. The result is a single write and read message transfer instead of multiple transfers. This approach gives my application a response time of 2 seconds or less to perform the entire operation. Of course, the disadvantage of this approach is the time needed to write the server program.

The steps required to process an order are as follows:

Step 1	An order number is entered into the pc application.
Step 2	The order number is sent to the mainframe.
Step 3	The processing on the host includes: - retrievals against several datasets. - summarization of data. - processing and returning the data to the pc.

Example 1: Implementation Example - Order Processing System

Implentation Example - Client Processing

The client processing consists of a program that calls functions in a library file. The library (LIB) file is used to create a simpler API to the server. By storing the communications transport link functions in a LIB file, we create a much friendlier programming environment that saves us future program development resources.

Three sample function calls are listed below;

init_hp, **read_hp**, and **close_hp**:

```
1) int init_hp(void)
   {
   printf("\nPerforming init_host...");
    err = init_host("myprogname", "ppl.cfg");
    if(err)
     {
      printf("\n*** Error %d in communications setup ****", err);
      return(TRUE);
     }
   return(FALSE);
   }

2) int read_hp(void)
    {
     printf("\nReading database...");
    err = hp_read(order_no, &dbstatus, &pc_data));
     if(err)
     {
      printf("\n*** Error %d in communications setup ****", err);
      return(TRUE);
     }
```

```
        if(dbstatus.image_cc)
                 {
                  dberror(&dbstatus, msg_buffer, &msg_length);
                 }
                    else
                     {
                   printf("\norder_no=%.20s", pc_data.order_no);
                  }
                 return(FALSE);
                }

            3) if (close_host( )) /* close communications link */
                printf ("ERROR - cannot close host connection.\n");
```

The init_host call in the init_hp function sends the name of our server program and name of the configuration file to the LIB file. The call returns an error if unsuccessful.

The read_hp function issues a call to the hp_read library function passing in the order number. If successful the data from the server is returned to the pc_data array. The Image status is also returned and is used to check for any database errors occurring on the host.

The close_host stops the server programs and closes the communications link.

Implementation Example - API processing

```
            1) int init_host(char *userhost, char *config_file)
               {
                int rtn_cc = 0;
                rtn_cc = GETSRV();
                if (rtn_cc != CS_SUCCESS)
                 {
                  return (rtn_cc+1000);
                 }
                parm_buf.CS_COMMAND = CS_CONNECT_CMD;
                parm_buf.CS_BUFFER = config_file;
                parm_buf.CS_BUFLEN  = strlen(parm_buf.CS_BUFFER);
                rtn_cc = CALLSRV(far_parm_ptr);
                if (rtn_cc != CS_SUCCESS)
                  {
                   return (rtn_cc + 2000);
                  }
                parm_buf.CS_COMMAND = CS_OPEN_CIRCUIT_CMD;
                parm_buf.CS_BUFFER  = userhost;
                parm_buf.CS_BUFLEN  = strlen(parm_buf.CS_BUFFER);
                rtn_cc = CALLSRV(far_parm_ptr);
                if (rtn_cc != CS_SUCCESS)
```

```
    {
     return (rtn_cc+3000);
    }
  }

2) int hp_read(char *order_number, DB_STATUS_ARRAY *db_status,
           char *pc_data)

  {
  int rtn_cc = 0;
  struct pc_buffer
   {
    int int_call; /* intrinsic call number */
    char order_no[20];
    } pcbuf;

  struct hp_buffer
   {
    DB_STATUS_ARRAY hp_status;
    HP_DATA_ARRAY host_data;
  } hpbuf;

  itoi (101, &pcbuf.int_call); /* intrinsic call number */
  memset(pcbuf.order_no,'\x20',sizeof(pcbuf.order_no));
  memcpy(pcbuf.order_no,order_number,strlen(order_number));

  /* PPL intrinsic to send buffer to host */
  parm_buf.CS_COMMAND = CS_SEND_WAIT_CMD;
  parm_buf.CS_BUFFER  = (void *)&pcbuf;
  parm_buf.CS_BUFLEN  = sizeof(pcbuf);

  rtn_cc = CALLSRV(far_parm_ptr);
  if (rtn_cc != CS_SUCCESS)
   {
    return (rtn_cc + 1000);
   }

  /* PPL intrinsic to wait for host response   */
  parm_buf.CS_COMMAND = CS_RECEIVE_WAIT_CMD;
  parm_buf.CS_BUFFER  = (void *)&hpbuf;
  parm_buf.CS_BUFLEN  = sizeof(hpbuf);
  rtn_cc = CALLSRV(far_parm_ptr);
  if (rtn_cc != CS_SUCCESS)
   {
    return (rtn_cc + 2000);
   }

  /* Set returned intrinsic variables */
  itoi(hpbuf.hp_status.image_cc, &db_status-image_cc);
  if(db_status-image_cc == 0)
   {
    memset(pc_data-order_no, '\x20', sizeof(pc_data-order_no));
    memcpy(pc_data-order_no, hpbuf.host_data.order_no,
          sizeof(pc_data-order_no));
```

```
// process the remainder of data from the hp here
        }
      }

    3)  int close_host(void)
        {
         parm_buf.CS_COMMAND = CS_DISCONNECT_CMD;
         return CALLSRV(far_parm_ptr);
        }
```

The header file to accompany this code is listed below:

```
typedef
 struct stat
  {
   int image_cc;
   int image_entry_len;
   long  image_recno;
   long  image_chain_len;
   long  image_bptr;
   long  image_fptr;
  }
  DB_STATUS_ARRAY;

typedef
 struct hpdata
  {
   char order_no[20];
   // define remainder of data here
  }
 HP_DATA_ARRAY;

typedef
 struct pcdata
  {
   char order_no[21];
  // define remainder of data here
  }
 PC_DATA_ARRAY;

DB_STATUS_ARRAY dbstatus;
HP_DATA_ARRAY hp_data;
PC_DATA_ARRAY pc_data;

/*** Function prototypes ***/
int init_host(char *, char *, char *);
int hp_read(char *, DB_STATUS_ARRAY *, PC_DATA_ARRAY *);
int close_host(void);
```

You must also include the PPL.H header file in your program.

The init_host calls the PPL function GETSRV to get the address of PPLPC.EXE from the device driver PPLCOM.SYS and stores it in application memory. The calls to CALLSRV (CS_CONNECT_CMD and CS_OPEN_CIRCUIT) get communication parameters from the configuration file and start the server programs. The PPL server program is called PPLHOST and will start the server program specified in the init_host call. A value of 1000, 2000, or 3000 is added to any communication errors occuring in the init_host function to help identify where the error occurred in the call.

Note: CALLSRV requires a far pointer to the address of your PPL parameter block.

The hp_read function uses a buffer to combine the order number with a transaction code used by the server to identify which transactions to perform. This buffer is sent to the server using the CALLSRV function. The CS_SEND_WAIT_CMD issues a "waited" call that causes the program to wait until the call is completed before continuing. The CS_RECEIVE_WAIT_CMD will wait for a response from the server. A successful call returns the requested data from the HP in the host_data buffer along with the status array. This data is copied into the pc_data buffer.

The itoi function is used to convert a 16 bit integer value to a format used by the HP or the PC. This is my own function, but is the equivalant of the ICONV toolkit function in PPL. If a communications error occurs, a value of 1000 or 2000 is added to the error code. This helps us determine where the error happened.

The close_host function stops the server programs and closes the communications link.

Implementation Example - Server processing

The server (host) program simply needs to access two IPC message files on the HP to exchange PPL messages with the pc. The server program uses the transaction code that I send in the beginning of the buffer to determine which operation it will perform.

The steps that the server program performs are as follows:

1.) Open a message file to read pc data and one to write data.
2.) Perform database open calls.
3.) Read a record from the message file.
4.) Determine the operation from the transaction code.
5.) Perform host functions associated with the transaction such as gathering data and loading a buffer to pass to the pc.
6.) Write a record to the outgoing message file containing the data to pass down to the pc.
7.) Wait for the next record to be sent from the pc.
8.) When the signal to shutdown PPL is sent (rec type = 2), close files and end program run.

WRQ sends good examples of server programs that illustrate these basic functions. See your documentation for current examples to get started with. The setup requirements on the PC include adding the PPLCOM.SYS device driver to your config.sys file and loading PPLPC.EXE into memory prior to running your program.

D.) Performance Tips and Advanced Features

Performance -

1.) Use minimal buffers in your transactions.

    The single most important performance consideration is to keep the size of the buffers that you pass at a minimum. Pass only the data that you require on both platforms. The messaging feature of PPL allows a maximum buffer size of 2048 bytes available for user data. If you have 2000 bytes of data, for example, you will see better performance by passing it all in one transaction versus breaking it down into smaller parts over multiple transactions.

    If your program only uses part of a dataset record, now is the time to start specifying partial lists in your DBGET calls. Of course, if you are doing a serial read on the host, you will want to send only the records you select to the PC.

2.) Combine intrinsic calls into one PPL transaction.

    If you write your own server program, you can customize the processing on the host and create your own intrinsics or intrinsic modes. For example, instead of issuing DBLOCK, DBPUT, and

DBUNLOCK calls separately, you could create a new DBPUT mode that would cause your server to issue all three calls. Or you may want to combine a DBFIND and DBGET operation. Whenever possible, create your own customized functions like my example program that perform multiple intrinsic calls on the server.

3.) **Use PPLHOSTP for your server program.**

If PM capability is a problem on your system you may have to use PPLHOST. In this case, set PPL-ACTIVITY-TIMER in your configuration file to a value of 1. A higher value will cause your transactions to wait for the next packet to be sent by PPL before executing.

4.) **Optimize your server data.**

Now is a good time to analyse your database designs. A serial read through a large dataset will be devestating to your client server application. Give your users direct access to their data.

5.) **Distribute your data between platforms.**

You may have some static data that is infrequently updated or data that can permanantly reside on the PC. If this is the case, you may want to look into using a database on the PC for some data storage. This improves the response time and allows you to further take advantage of PC power.

Advanced Features

1.) **Conversion to Windows**

The PPL for Windows Version 1 will soon be released. Although the current version includes Windows support, it is worth waiting for the new version before starting your Windows applications. The PPL library has been replaced with a Dynamic Link Library (DLL). The main advantage with a DLL is that the functions are linked into your program at run time instead of at link time with a library (static link). This allows programs to share the same functions in memory and yields smaller executable files. The new version has greatly improved error handling along with many other features geared towards Windows. Support for using Reflections script files is included for starting Reflections in the background and establishing host sessions. This feature removes the burden of writing numerous "transmit" calls in your programs.

It is a fairly easy task to convert your library to a Windows library. The init_host function will now perform the following PPL calls:

- pplInitializeHostSession	(establish host session)
- pplStartServer	(start PPL Windows server program)
- pplConnect	(start PPL host program and open connection)
- pplGetFreeCircuit	(obtain circuit handle)
- pplOpenCircuit	(opens circuit)

The messaging calls used in your other functions will also change slightly. The "send" message calls may include the functions below:

```
- lpMsg = GlobalLock(hPPLMem);    (set up global memory lock)
_fmemcpy(lpMsg, outbuf, len);     (copy our buffer into global
                                           memory)
GlobalUnlock(hPPLMem);            (unlock memory)
err=pplWriteMessage(hContext, hCircuit, hPPLMem,len, TRUE, &pid,
NULL, 0);
```

The call above replaces our CS_SEND_WAIT_CMD call with a Windows call to send a message to the host. Please refer to the PPL for Windows manual for a detailed explaination of the above calls.

If you've used the approach of writing your own API, most of your changes in this conversion are done at the library level with mimimal changes needed in your programs. The server program will not be affected by this conversion unless you change some of your functionality.

2.) Convert your library to a DLL.

If your functions reside in a DLL, you will be able to easily access them using some of the Windows application builders like Objectvision (Borland), Visual Basic (Microsoft), or Actor. You also gain all the other benefits with a DLL.

A library assumes that the code segment register (CS) is in the same area as the data segment register (DS) or CS=DS. This is not true for DLLs. A DLL has its own DS area. This has quite an impact programmatically. Quick tips include defining your functions as FAR PASCAL, defining variables as STATIC in the DLL,and only using far pointers. For a detailed explaination of creating a DLL, I found *Programming Windows* by Charles Petzold an invaluable reference.

3.) Use the PPL callback mechanism in Windows.

"Waited" calls take control away from Windows and force a "wait state" until the call completes. This removes the multi-tasking feature of Windows and can disrupt other applications requiring some processing time. The callback function will let you initiate the call to PPL and then periodically check the status of the call. This also provides you with a safeguard to prevent the program from hanging due to a communications problem.

Client server applications introduce the problem of transactions failing due to problems in the environment such as cables being disconnected, phone line disconnects, program timeouts, buffer overflows, and many other situations. Callback functions on a "non-waited" call allow you to detect a problem and either try to correct it or close the application in an orderly fashion.

PPL for Windows ships with a demo called WXFER.C that will provide with a good example of how to get started using this feature.

Typically, your client server applications should provide fast response time, but if you find an example where you want to initiate a server process and then return control to the screen, use nowait calls with the callback functions.

Another good example of callback processing would be to simulate a "prefetching" environment similar to the HP. You may want to transfer data in the "background" while the user previews the data already in memory. Be careful not to overwrite the amount of memory that you globally allocated!

4.) User Picklists.

A nice feature to include for your users is the ability to select or qualify a number of records on the server and then view a small portion of several records before transferring the entire record. This gives the user the chance to preview portions of several records at a glance before selecting the exact record to view in detail.

This feature can be accomplished by designing an intrinsic that qualifies records based on the user's selection criteria, and then downloads one or more fields from the list in "page blocks". The user can then page forward or backward and see a small block of records at a glance. Using the Windows LISTBOX control to display the data will produce some automatic features such as horizontal and vertical scrolling. The user can select the desired line and transfer the entire record to the client for detail processing.

5.) **Share the client data with other Windows applications.**

You can also build Dynamic Data Exchange (DDE) or Object Linking and Embedding (OLE) support into your client server application. This will open up a whole new environment for your users to share the imported server data with other Windows applications.

Summary

The introduction of many new software packages recently has made the conversion to client server based applications much more attainable for the HP data processing shop. The popularity of Windows is quickly spreading among our users and we are faced with more demands in our applications today than ever before.

The selection of software to achieve the results we desire is getting more and more complex. I found PPL to be a great introduction into this environment because the initial investment is mimimal and the portability allows growth onto other platforms. Applications are not restricted to their present platform. Combining PPL with a portable language such as C is a first step toward open systems compliance. I think this concept will be a cost effective approach in a time when hardware is rapidly changing and lowering the average life of a system.

Good luck with your new environment!

#3801

HP3000 OPEN DISTRIBUTED OLTP

RICK BARTLETT

HEWLETT-PACKARD COMPANY
19447 PRUNERIDGE AVE, MS 47UH
CUPERTINO, CA 95014

PHONE: 408-447-6455

HP3000 OPEN DISTRIBUTED OLTP

ABSTRACT

Business On-Line Transaction Processing (OLTP) needs for many HP3000 MPE iX customers will be undergoing a shift in the next couple of years towards distributed computing paradigms, such as Client/Server. These needs will be based on the realized benefits of distributed computing which include wider availability of resources and more efficient utilization of resources in a networked environment, as well as the productivity benefits offered by desktop user interfaces. The emergence of key interoperability and portability standards, as well the blossoming of the "Open" movement, have resulted in increase momentum for business solutions by distributing applications.

In planning to keep pace with the shift to higher levels of cooperative computing, MPE/iX is incorporating new technologies and improving it's infrastructure in order to maintain leadership in commercial OLTP as well as to extend it to the Open Distributed Computing arena. New capabilities are coming to the HP3000, including Transarc's Encina, an open transaction processing architecture, a requirement for distributed OLTP applications. Encina will be built on OSF'S Distributed Computing Environment (DCE), which provides a set of open services for building distributed applications. Both of these capabilities will be utilizing the services of the MPE iX Connection Architecture, which provides much more efficient support for an MPE/iX Server which may be servicing large numbers of Clients.

Discussed will be the business needs that the MPE iX Distributed OLTP program addresses, product technology overviews of the Transarc's Encina, DCE, and Connection Architecture, and topological considerations such as Client/Server, Terminal/Server as they apply to these technologies.

BUSINESS OVERVIEW

Today's business environment presents a number of information management challenges as companies are increasingly becoming more distributed in order to compete in a global economic environment. Worldwide distribution of operations and inter-enterprise partnerships create demands requiring information flow more easily between previously autonomous business units.

Continued growth in demand for applications, easier to use applications, and increased services for the end-users is expected. Users need access to data regardless of it's location in an enterprise. The business environment today is characterized by rapid change, requiring an information technology infrastructure that can respond to an ever changing set of needs in a timely manner. The ability to manage and deliver information in whatever form it is needed is becoming increasingly strategic to the enterprise.

Rapid technological advances have presented additional opportunities for business solutions. Dramatic increases in processor performance, network capacities, storage capacities, and graphics capabilities have opened the door for a new range of solutions that can address a new order of needs and benefits.

As seen from the HP3000 customer base, business success is directly related to at least these factors:

1) Compress Time to Action: The ability to respond to changes is key to competitiveness. This is best accomplished by facilitating access to enterprise-wide data. Information access enables otherwise isolated (but related) data to be synergized offering added value and time savings to resolve perhaps a multi-step problem.

2) Improve Operational Efficiency: Organizations are changing rapidly in response to competitive pressures, geographic considerations, and a host of other reasons. In order for information technology to be effective in this type of environment, a flexible environment must be maintained. In addition, a key issue for IT organizations is to be focused on doing more with less. This involves minimizing user training, rapid deployment of new applications, and reducing system administration costs.

3) Improve Customer Service: Customer service is becoming much more of a key differentiator for businesses. Information access will be key to providing improved customer service. Information such as order status and delivery time, opportunistic selling options, and generally a more

one-stop/one-person service to address a customer's needs requires information access to a wide range of enterprise (and inter-enterprise) information. Customer service is that property of a business transaction that benefits most from fast access to data.

Many businesses are impeded from utilizing their current information technology to contribute towards these goals due to a rigid application set and topology. Many IT organizations are using dated technologies, which operate comfortably in the stability in the era they were deployed. Today's business environment is based on change and change management. Businesses cannot survive with the long lead times required to modify (and certainly to create) existing applications in these environments.

The following is a list of the urgent needs that our customer base is really focused on improving:

- Accurately setting customer expectations on orders, prices, and delivery dates
- Forecasting more accurately
- Getting order confirmation instantly
- Entering order accurately only once
- Getting customer order history
- Speeding credit approval
- Tracking order status
- Automating ordering
- Delivering On-Time
- Delivering accurately against order
- Minimizing delivery expense
- Optimizing delivery
- Maintaining one logically consistent copy of data

While these items are generally attributed to manufacturing and distribution, many of the aspects of these items have analogs in any other industry (insurance claims processing for example).

All of these needs that are typical of any commercial environment can be improved by uncloaking the information that is usually already available but not easily accessible within the enterprise or even between enterprises (such is the case when you reserve an airline flight and they now ask if you would like to reserve a car as well). Information technology is key to resolving these strategic corporate issues. All of the items in the preceding list are business transactions or components of larger business transactions, and if they currently are not handled on-line, may need to be in order to to be fully addressed.

ENTERPRISE-WIDE DISTRIBUTED OLTP

The success of the HP3000 is based on it's ability to deliver On-Line Transaction Processing (OLTP) in which data integrity and reliability are core requirements. In an enterprise, with multiple data and processing sources, an extension of OLTP is required in order to escape the confines of a single system in the context of a business transaction. Enterprise-wide OLTP enables the enterprise to operate across geographical and organizational boundaries. Without enterprise-wide OLTP, there are discrete processes that cannot be coordinated.

For example, in a large business, there may be a sales call to a sales representative who takes an order using a price list to ascertain the total price to the customer. The order entry clerk then gives the order to a data entry clerk who enters the order in the order entry system. The order may then be faxed to one or more inventory centers to process and ship. All of this data is then fed to a mainframe computer for nightly processing to produce customer billing and other decision support information. In this scenario, there is built-in latency as many manual steps are introduced, as well as an increase in the chance for error (lost or duplicated data).

Utilizing key core technologies, enterprise-wide OLTP eliminates the possibility for lost or duplicated data, introduces a new standard for integrity within the business information processes, and provides the basis for improved customer service (both for internal and external customers). Enterprise-wide OLTP in the context of the previous scenario would enable a single, multi-windowed display used by a knowledge worker to have access to current on-line pricing information, order history, order status, and order delivery time. The distributed application would then be able to lock in the current price, place the order, check inventory, and schedule shipment, and update the corporate financials within seconds, within the context of a single transaction that would guarantee that all data stores are updated.

To provide truly enterprise-wide OLTP, there must be an infrastructure for hiding the complexity of multiple heterogeneous computing environments. An abstraction of the mass of individual networked systems components must be applied as well as a set of services provided to enable the view of an integrated distributed computing environment. In addition, there must also be the capability to enable business transactions to operate within this distributed environment, where a transaction may span multiple systems and update multiple databases (or other recoverable resources).

There is a set of requirements that must be fulfilled in order for this distributed transaction processing environment to be successful:

- o Standards based API's for portability
- o Interoperability across multiple databases and file types
- o Interoperability across multiple vendors platforms
- o Data integrity and recovery across multiple systems
- o Good OLTP performance across multiple systems
- o PC, terminal, and workstation connectivity
- o Support for large numbers of users

The HP3000 is well positioned to address business needs in this new global distributed environment. Building on our base of open systems interfaces for interoperability and portability, commercial OLTP, and comprehensive networking support, the HP3000 has incorporated new technologies which provides businesses the enablers for approaching this new distributed OLTP paradigm.

HP3000 CORE DISTRIBUTED OLTP TECHNOLOGIES

Several new technologies are positioned to propel the HP3000 into a distributed computing environment which can address enterprise-wide OLTP. They include the Open System's Foundation Distributed Computing Environment (OSF DCE), the Encina distributed transaction environment from Transarc, and the Connection Architecture product.

OSF DISTRIBUTED COMPUTING ENVIRONMENT

The Open System Foundation (OSF) is an industry consortium consisting of several hundred organizations worldwide, including most of the major computer vendors (HP, DEC, IBM, Apple, Novell, Microsoft, etc.), as well as a large number of major industries (Boeing, CitiCorp, American Express, Exxon, etc.) and universities (Stanford, CMU, MIT, U Mass, etc). The OSF has selected a set of technologies in an open process which addresses interoperability in a distributed environment, which has been identified as the greatest need in the industry. This set of technologies is the Distributed Computing Environment (DCE). Hewlett-Packard has licensed this technology from the OSF and is integrating DCE into the HP3000 MPE/iX operating environment.

DCE is a comprehensive, integrated set of services that support the development, use and maintenance of distributed applications. DCE provides the software foundation for transparent integration of independent, heterogeneous computers. HP considers DCE the core competency for distributed computing for the 90's. DCE is riding a wave of support in the industry with the following list of attributes:

- o Consistency for easier multi-vendor network implementations
- o Facilitation of client/server applications

- o Industry Standards to provide portability
- o Shields developer from different hardware and operating systems dependencies
- o Has capabilities for secure, manageable, flexible and synchronized distributed applications
- o Core technology for Transarc's Encina distributed transaction processing environment

DCE is the foundation for distributed OLTP applications. DCE provides the services that enable a network of individual systems to be viewed as a repository of computing and information services which span physical system boundaries by providing naming/directory services, resource sharing and network-wide security.

The DCE is comprised of a set of integrated services. The core services for DCE are the following:

- o Remote Procedure Call (RPC) - enables individual application procedures to be able to execute remotely in the network, resulting in distributed application execution.
- o Threads - portable facilities providing support for concurrent programming.
- o Directory Service - provides a single naming model to enable location transparency for named objects in network space.
- o Distributed Time Service - provides accurate and consistent timing for distributed applications.
- o Security Service - provides authentication, authorization, and user account management for distributed environments

The DCE RPC is a set of portable software development tools implementing an open architecture for network computing. The DCE RPC is based on the premise of extending the local procedure call model to one that can execute individual application procedures remotely on any system in a network. DCE RPC provides presentation services that mask differences in data representation between different machine architectures. DCE RPC is also integrated with the rest of the DCE services resulting in these benefits:

- o Secure RPC - several degrees of secure communication is provided by the integration of RPC with the Security Service.
- o Location Transparency - RPC is integrated with the Naming service which provides location transparent naming for locating servers. A remote procedure server can be moved to another location within the network (to a faster CPU for example) and clients calling the procedure are not aware of the change.
- o Network Independence - RPC is implemented in a network transport independent fashion. RPC's run over a number of network transports (TCP/IP, OSI, etc.).

- o Threads integration - enables clients to interact concurrently with multiple servers and enables servers to handle multiple client requests in parallel.

The DCE RPC provides a set of tools for defining interfaces (procedure definitions for remote procedure calls) which look like a local procedure call definition. The interface contains minor additions for describing variable modes (in/out/inout), binding options, etc. This interface definition is then compiled through a DCE IDL (Interface Definition Language) compiler, resulting in a client stub, server stub, and header file. The stubs are typically conventional programming language code (like C). The client stub presents to the client what looks like a local procedure call, but which actually contains calls to the RPC runtime code to pack parameters, perform binding, etc. The "procedure call" is then forwarded to the server, where the RPC runtime on the server side invokes the server stub to unpack parameters, perform and data conversions, and then call the application code as if the client program had called it directly.

The initial release of the DCE IDL compiler generates only C language compliant stubs. As COBOL is the predominant programming language in the commercial sector at this time, there is a widely recognized need for COBOL language support by DCE. There are a number of methods and tools available that allow COBOL to DCE C stub interactions to take place. Additionally, there are a number of COBOL DCE IDL compiler's in development at this time, which will allow for tighter integration of existing and new COBOL applications into the DCE RPC environment.

The DCE RPC allows the code that operates on data to be located on the same machine as the data. This results in improved and optimized performance, and as compared to more traditional remote data access techniques (such as RDBA) can reduce networking traffic. Additionally, the RPC programming model is much more intuitive and consistent with current structured programming methodologies.

Threads is a DCE service which enables applications in a distributed environment to take advantage of the inherent parallelism that exists in distributed computing environments. The Threads Service provides the operations necessary to create and control multiple threads of execution within the context of a single process. Threads provide a light-weight execution environment, and share global data within the context of a process. Threads are used extensively by most of the DCE services. The DCE supplies user user-space version of threads that can be used unless the resident operating system provides a kernel-level threads package. Such a capability has been developed on the HP3000, which results in competitive performance advantage (especially apparent on multi-processing systems) as compared to systems that run DCE applications with user-space threads.

Threads are the basis for efficient use of resources and optimum utilization of processing power in a multiprocessing system. For example, with a client application that may be accessing multiple services from multiple servers, it is desirable to make a RPC request to one service, and instead of serializing until that procedure call returns, be able to create another execution thread to make an RPC request to the next service in parallel. In addition, it may improve response time to create another parallel execution thread within the process to handle user input. In the server environment, for a server that is handling multiple requests from a number of clients, that server would benefit from creating multiple execution threads to handle concurrent client requests for service. This concurrency provides improved performance where locking of resources is occurring, with excellent prognosis for performance advantages in multiprocessing environments with kernel-level threads.

Directory Service provides a single naming model to locate name resources in a distributed environment. Named resources, such as servers, files, disks, print queues, etc. can be located transparently by utilizing the directory service. The DCE directory service is comprised of three components:

- o Cell Directory Service (CDS): The CDS provides the naming service for resources that reside in a DCE Cell. A DCE cell is a logical grouping of DCE nodes that are to be administered together. A DCE cell is typically defined based on organizational or geographical boundaries if there is to be more than one cell. CDS provides an optimized naming service for local access by caching name requests. CDS also can be replicated within a cell for high availability.

- o Global Directory Service (GDS): With multiple cell DCE configurations (multiple organizations or companies), the Global Directory Service is available to provide access to naming information outside of the local DCE Cell. The Global Directory Service is distributed and replicated. It behaves as a higher level member of a naming hierarchy relative to CDS. GDS is based on X.500, which is a worldwide naming standard. In addition, DCE supports the use of DNS (domain name service) as a global enterprise (inter-cell) name service, which is widely used in the internet community.

- o Global Directory Agent (GDA): The GDA is used to interface between CDS and the global directory service X.500 or DNS.

The Directory Service is implemented on top of the DCE RPC, which provides a mechanism for secure communication, as well as the ability to interoperate between heterogeneous computers over a variety of network transports.

Time Service regulates and sychronizes system clocks in the computer network, providing consistent time for distributed applications. This feature is often undervalued, but in a distributed application it is critical that clocks where portions of the application are running be in synch in order to present a single-system image. The Time Service consists of time servers which periodically synchronize the clocks on different systems. These servers also have a notion of understanding which clock sources are closest to the correct time, as well as understanding the degree of inaccuracy for any time source (perhaps due to networking latency). The Time Service provides an interface to an External Time Provider which usually provides a very accurate time source (atomic clock, radio service, etc).

The Security Service provides a distributed environment with the protection that is necessary with the introduction of a highly networked environment utilizing distributed resources. Also, a consistent set of security semantics is necessary to hide the inconsistencies of native operating system security within the heterogeneous computer environment. The Security Service provides the distributed environment with authentication, authorization, and user account management services.

DCE security service provides robust authentication. Authentication enables two processes on different systems to be sure of each other's identity. The authentication service is based on Kerberos system from MIT, and utilizes time-synchronization and private-key encryption/decryption techniques.

After authentication takes place, a user must be authorized to use resources that the user may be requesting. This authorization is provided by the Privilege Service. The privilege service provides in a secure fashion, information that a server needs to know in order to grant access by the user to that server.

In a distributed environment, the concept of a networked user requires a place to store information about that user (unique user name, passwords, account information, rights, policies, etc). A Registry Service provides a replicated service that maintains a secure database of such information.

The security service makes use of and is used by DCE RPC to provide a secure means of communication. In addition to the authorization and authentication security, the security service provides RPC's with increasing degree's of protection on the data within the RPC itself:

- o No Authentication
- o Authenticate on first call to server
- o Authenticate on each call to server
- o Authenticate on each packet between client and server
- o Ensure data integrity by checksumming
- o Encrypt data using DES

The DCE core services provide in an integrated and consistent fashion, the core requirements for an infrastructure to build distributed applications. Rather than develop these services, utilization of the DCE frees up application development resources to spend time on the business related portion of the application.

BUSINESS Application Programming Interfaces (API'S)

The DCE Remote Procedure Call coupled with the other DCE services provides an enabler for what we refer to as "Business API's". With DCE, an enterprise generally includes fairly autonomous units who want control over their computing resources and data stores. For example, the manufacturing entity may have different computing resources than the sales/marketing organization. In order to facilitate access to each other's data, each organization could export a set of higher level DCE RPC's (Business API's) to interface to their organization (Sales/Marketing: Customer_Info, Order_History, Sales_Data, Today's_Sales, Manufacturing: Update_Inventory, Items_In_Stock, Get_Backlog).

The accumulation of these sets of business API's from the organizations within the business would result in conceptual Business API for the enterprise. This API set would be the building blocks for business applications, and would generally be distributed applications. In this way, information can be accessed and updated, and local autonomy is preserved. The local organization can rearrange, add (perhaps specialized servers) or delete, computing and data storage resources without affecting the interfaces from which the business applications would be coded to. This results in a scalable architecture for distributed applications.

We believe that the business API is a powerful concept, and DCE is the enabling technology.

DCE provides the services for the distribution of applications and resources within a networked environment. DCE also provides the base for transactional services to provide the integrity and reliability that is required for distributing business transactions.

ENCINA DISTRIBUTED TRANSACTION PROCESSING ENVIRONMENT

The Transarc Encina technology combined with core MPE/iX OLTP capabilities provides high performance distributed heterogeneous data access with complete data integrity, recoverability and control. The Encina/iX product provides ease of development and management of complex transaction processing (TP) applications in a multivendor client/server OLTP environment.

The most important attribute of a transaction processing system is transactional integrity. The transaction processing system must be able to handle many concurrent users accessing services. It must also be able to provide robust recovery from failures.

Transactions possess the following properties, referred to as the ACID properties:

- o ATOMICITY: all of the operations of a work unit are performed or none are performed at all.
- o CONSISTENCY: operations are performed accurately with respect to application semantics.
- o ISOLATION: the partial results of a work unit are not accessible.
- o DURABILITY: the completed unit of work has permanent effects which are insensitive to failures.

One of the major roles of a distributed transaction processing system is to coordinate consensus between multiple parties that are participating in a transaction. In the distributed transaction processing environment, this is accomplished with a protocol called two-phase commit. The two-phase commit protocol is a multi-step commitment protocol to ensure that the Atomicity and Durability properties are applied to a transaction which encompasses multiple systems and databases. A commitment coordinator is responsible for arriving at a go/no-go decision for the global transaction. The commit coordinator, with Encina, need not be the initiator of the global transaction. This enables appropriate placement (high availability system) of transaction coordination to a server system even when the global transaction was initiated from a workstation or PC. This capability also allows a client to have a subset of Encina code rather than the complete package, making the client that much more viable in a distributed TP environment.

An important aspect of a successful transaction monitor is that it conform with the X/Open DTP model. X/Open is an independent worldwide open systems organization support by most of the world's largest information systems suppliers, user organizations and software companies. X/Open's DTP model is a software architecture that allows multiple application programs to share multiple resources provided by multiple resource managers (databases, communication systems), coordinating their work into atomic transactions enterprise-wide. The X/Open DTP specified open API's and system level intrinsics that provide for:

- o portability of applications
- o interchange of transaction managers and resource managers (databases)
- o interoperability by different resource managers (databases) and transaction managers

The X/Open DTP defines the transaction semantics and syntax that is used (TX interface) by an application to control the transaction (begin, end, commit, rollback). In addition the XA interface interfaces between transaction monitors and resource managers (databases and communication managers) to structure the work of the resource managers. The XA interface also enables the transaction monitor to coordinate global transaction completion and recovery. A benefit of the X/Open DTP model is that the interface of application to resource manager (database) remains unchanged. This makes it much easier to incorporate existing applications into a distributed transactional environment. The X/Open DTP model also is defining a communication interface which attempts to standardize the way on application communicates with another. The CM interface has not yet been finalized, but will likely include Transactional RPC (discussed later), Peer-Peer, and Client/Server interfaces.

The business benefit of choosing an X/Open DTP compliant transaction processing solution is availability and integration of a number of transaction processing components that can participate in distributed transactions. Most major database vendors are or will be X/Open XA compliant. Also, portability between different transaction processing systems is guaranteed by using the same TX interface and communication interface (RPC, Peer-Peer, Client/Server).

Transarc's Encina is HP's Strategic Transaction Processing monitor (strategic for HP3000 and HP9000 platforms). Encina is built in a modular fashion, such that vendors can integrate to different degrees within their operating environment to add value. Interfaces between all major modules are open. Encina is also designed with a commitment to standards; it is built on and utilizes the services of OSF's DCE, and adheres to the X/Open Distributed Transaction Processing model (to be discussed later). It also supports transaction services to SNA peer-peer environments using LU6.2 sync-level 2 services.

Encina is a forward looking industry strength product with a strong research base. Advanced capabilities such as transactional

RPC's, nested transactions, and multi-threading translate into both current and future opportunities for a powerful and high-performance distributed transaction processing base. In addition, Encina has been endorsed by many of the leading computer and database vendors (HP, IBM. NEC, Stratus, Sybase, Oracle, Informix, etc.'. Hewlett-Packard has licensed Encina from Transarc for integration into the HP3000 operating environment.

The Encina/iX architecture consists of five primary modules which expand the DCE foundation to include services that support distributed transaction processing and recovery:

- o Base Development Kit (BDE): is a portability layer that interfaces to the host operating system. The BDE isolates system dependencies from the Encina product suite.

- o TP Monitor: the TP Monitor includes additional services for configuration, administration, security, performance assurance, and tool integration support. The TP Monitor provides a development, execution, and administration environment for managing complex Transaction Processing (TP) applications.

- o Tool Kit: the Tool Kit contains a set of modular components that support the development of distributed OLTP applications. The modules provide support for two-phase commit, transactional RPC, locking, recovery, logging, and X/Open XA interface.

- o Transactional C: provides additional language constructs to the C programming language to simplify application development in regards to transaction demarcation, concurrency control, and exception handling.

  An example of a transaction with (approximation of) Transactional C follows for a debit/credit scenario:

```
Transaction
   Read (Account1, Account2, Password, Transfer_Amount)
   if Valid (Account1, Password)
      { if Transfer_Amount < Deposit (Amount1)
         { Debit (Account1, Transfer_Amount)
           Credit (Account2, Transfer_Amount) }
        else
         { Abort_Reason = "Insufficient Funds"
           Abort_Transaction } }
   else
      { Abort_Reason = "Incorrect Password"
        Abort_Transaction }
On Abort
```

```
      printf ("Transaction not processed ", Abort_Reason)
   On Commit
      Write_Journal (Account1, Account2, Transfer_Amount)
```

Note that in the above examples with DCE and Encina, any of these procedure calls (especially Debit and Credit) may be on different machines accessing different databases. This is transparent to the application developer.

The Encina product builds upon the DCE RPC semantics by providing a transactional RPC. The transactional RPC exhibits the ACID properties with respect to a remote computation. Unlike a standard DCE RPC, a Transactional RPC (TRPC) when it fails, rolls back all work done on behalf of a global transaction. The greatest beauty of the transactional RPC, is that it looks just like a DCE RPC in terms of definition and execution. Additionally, the TRPC retains inherits all of the services that come with DCE RPC (naming, security, interface definition).

The Encina TP Monitor contains the provides a development, execution, and administration environment for successfully creating and deploying distributed OLTP applications:

o Development Environment Features: API's and tools necessary for distributed transaction processing application development.

- Transaction Demarcation & Control API's
- Front-End tools integration (including JYACC JAM, other 4GLs, MOTIF, CASE tools)
- Problem determination with diagnostic reporting and audit trail info

o Execution Environment Features: services to optimize performance and to insure high availability & security.

- Transparent binding: provides load balancing dynamically across multiple application servers as well as rerouting work from a downed server to one that is active.
- Priority scheduling of servers
- Guaranteed deferred execution of tasks (Recoverable Queuing service)
- Protected execution environment with access control lists as well as isolating servers to a client during access.

o Administrative Environment Features: interfaces and tools for system management to configure and manage distributed OLTP applications.

- Comprehensive view of the entire distributed TP system

- Monitoring of active clients, server availability and load, client authorization, centrally logged error conditions, auditing information.
- Management functions: startup, shutdown, and migration of servers in the distributed environment.
- Replicated administrative and configuration tools

Encina/iX enables complex business transactions to operate within a network of DCE (and SNA LU6.2 sync-level 2) nodes providing full data integrity and recoverability. Transactions can be supported across multiple resources (databases) on a single SPU, as well as across multiple SPU's. Encina is an open environment, with broad host platform and database support.

The Encina/iX product builds on top of DCE by providing transactional integrity while leveraging from the RPC paradigm and services provided by DCE. The Encina/iX also further strengthens the "Business API" concept by providing transactional integrity and recoverability to applications that are constructed using these API's (Transactional DCE RPC's).

CONNECTION ARCHITECTURE

The value of the MPE/iX Connection Architecture is in providing an environment for efficiently managing large numbers of connections required by high-end systems as well as for HP3000's of all sizes that are in client and server configurations. The connection architecture introduces a new environment outside of current MPE/iX job/session environment which provides much better utilization of resources for HP3000 systems in standalone and distributed configurations.

Before Connection Architecture, each client (terminal or remote session) required at least three system processes (CI, JSMAIN, application process) to logon to the system and run an application. A server (usually implemented as a background Job) also required at least three processes to run. In both of these cases, there are considerable resources consumed with each process (tables, data segments, etc). Quite often, both the clients and the servers exist only to run an application to completion and then terminate. In these cases, the CI and JSMAIN processes are not required to be involve to execute the task. The MPE/iX Connection Architecture provides the mechanisms for eliminating this excess overhead.

The benefits of this new connection environment include:

- o Much higher HP3000 maximum connection client (terminal user) capacity (up to 5000 initially).

- o Standard architecture to eliminate the need for ad-hoc connection methods.
- o Reduced resource utilization for same number of clients resulting in reduced cost/user.
- o Improved software resiliency over current connection methods.
- o Easy to use interface and management facility.
- o Efficient client/server architecture for DCE and Encina environments.

The Connection Architecture enables clients to logon to the system and be deposited into the application (as if they had a logon UDC), in such a way that existing terminal users should be able to be migrated to this environment easily. With connection architecture, servers no longer require a background Job, but are created as a Connection Architecture environment and managed within that scope.

In addition to the light-weight operating environment, connection architecture provides the demand creation of a list of resources that are infrequently referenced (JDT, temporary file directory, etc.). This change to dynamic creation provides consistent functionality with the benefit of saving system resources.

The Connection Architecture Environment space is integrated into the core of the MPE/iX operating environment. In addition to a functionally complete set of primitives for managing the environment life cycle and information requests, MPE/iX commands are extended to reflect the new Environment aspects where appropriate. Environments are able to have a class attribute for identification and grouping of operations. This is particularly useful on systems with large numbers of clients, where SHOWJOB does not offer the granularity resulting in potentially thousands of lines of output to look for one job or session.

The Connection Architecture is especially valuable when used by DCE and Encina distributed applications. Clients and Servers fit very naturally into this environment, resulting in efficient, consistent, and more manageable distributed applications.

HP3000 OPEN OLTP SERVER PROGRAM

The HP3000 Open OLTP Server Program is the vehicle for delivering the capabilities of Open Distributed Transaction Processing. The three main technology components are OSF DCE, Transarc's Encina, and the MPE/iX Connection Architecture. The main objective of the program is produce a solution which integrates these core technologies into a competitive solution that addresses the business needs of our evolving customer base. In addition to the core technologies, there are additional major program components not described in this document. These components include consulting, support, and application development (tools, 4GLs, CASE), which are key to bringing customers onto the core technologies and into the distributed OLTP computing environment.

The Open OLTP Server Program is strategic for the HP3000 platform. HP's Commercial Systems Division is committed to ensure that the program's deliverables live up to the high standards for quality and completeness that HP3000 customers expect. The program has developed a comprehensive solution with technologies from both internal R&D (Connection Architecture, kernel-level threads) as well as with external industry leading technology suppliers (OSF's DCE, Transarc's Encina) to deliver the business benefits of distributed transaction processing. This solution builds on the traditional commercial OLTP strengths provided today by the HP3000 MPE/iX open operating environment with coupled with HP's PA-RISC hardware platforms.

OSF DCE is a trademark of the Open Software Foundation, Inc. in the USA and other countries.

Encina is a trademark of Transarc Corporation

PAPER #3802

CONFIGURING YOUR DISKS FOR HIGH AVAILABILITY

Dave Beasley
Hewlett-Packard Company
19111 Pruneridge Ave
Cupertino, CA 95014
408-447-5864

As the number of users on MPE/iX systems continues to grow, disk storage requirements are also continuing to rise. Additionally, there is an ever increasing demand for highly available systems. More and more of our customers are telling us that they simply cannot afford any significant downtime. With this in mind, it is very important for a system manager to have a good backup and recovery strategy; one that will minimize downtime and allow the system to be available to the end users as much as possible. There are many factors involved in planning a good backup and recovery strategy. You must consider questions such as how should the data be backed up (by application?, partial backups?, full backups?, etc), how often should it be backed up, and should you be doing some form of transaction or database logging? One question that may not be as obvious to ask, but is very important in planning a good recovery strategy, is how will I configure and manage my disk subsystem?

Regardless of how reliable HP disk drives are, they do fail on occasion, and of course if one does, it will happen on a Friday night before a holiday or during month end, right? Disk failures are never completely painless, but if you haven't done your homework and prepared properly, they can be your worst nightmare, resulting in significant downtime and/or lost data. This article will point out some ideas that you can use as you plan your disk configuration which will help to minimize or totally eliminate disruptive downtime in the event of a disk hardware problem.

First we will look at the benefits of using User Volume Sets which may significantly improve recovery time. Next, we'll learn about the features of HP's new C225XHA disk array products that will allow the system to continue normal processing during a large percentage of disk hardware problems. And lastly, we'll take a look at the Mirrored Disk product which will virtually eliminate any system downtime by using redundant hardware for the disk subsystem. You can select any one or more of these methods, depending on your own requirements.

USER VOLUME SETS, WHY NOT GIVE THEM A TRY?

For those of you with an MPE V/E background (or earlier), I'm here to tell you that User Volumes (i.e. Private Volumes in the old days) are no longer a dirty word! Volume Management on MPE/iX is a basic part of the MPE/iX operating system. That means it's free first of all! You do not have to pay anything extra to use it. Secondly, Volume Mangement uses the same code to manage and access User Volume Sets as it does for the System's Volume set (MPEXL_SYSTEM_VOLUME_SET). So it's not a facility that has been 'bolted on' as was the case with MPE V/E's Private Volume Facility. This means that you will not suffer any performance degradation by taking advantage of it.

I'm not telling you this to 'run down' the MPE V/E Private Volumes Facility because I know that several customers have taken advantage of them very successfully. But, they were aware of the tradeoffs, particularly in the area of performance. However, I do know that there has been a negative perception to 'Private Volumes' in the past. So, what I'm trying to say, if you haven't figured it out by now, is that these 'fears' are no longer warranted and that every MPE/iX customer should seriously consider using User Volume Sets. The benefits more than outweigh the small effort required to set them up.

VOLUME MANAGEMENT BASICS

Volume Management allows you to partition your data into separate, individually managed sets. Your disk space is treated in terms of volumes, volume sets, and volume classes. A volume can be thought of as a single disk device or ldev. Each volume or ldev has its own volume label, disk free space map, file label table, and one or more extents of file data. A volume set is made up of one or more volumes. Volume classes can then be used (optionally) to further subset space within a volume set. Files can then be created or restored with a volume class restriction, providing greater flexibility for managing your disk storage. Volumes are allowed to have more than one volume class, so volume classes may overlap within a volume set. VOLUTIL is the utility to use when creating volume sets and volume classes.

One volume in the set is considered to be the master volume and the others are considered to be member volumes. The main difference between the master volume and a member volume is that the master volume contains the directory information about accounts, groups, and files associated with that volume set. (On the MPEXL_SYSTEM_VOLUME_SET, the files that make up the system directory are allowed to spread across all members of the System Volume Set. But on User Volume Sets, the volume's directory is restricted to the master volume). The master volume also contains the Volume Set Information Table (VSIT) for the volume set as well as the Transaction Management logfiles.

The MPEXL_SYSTEM_VOLUME_SET is the only required volume set on the system. It must be mounted in order for MPE/iX to function properly. It is created automatically when you first install MPE/iX. Ldev 1 is the master volume of this set. There must be enough disk space configured in the System Volume Set for all of MPE/iX's files, for transient space, and for Spool file space. I can't tell you precisely how many members (volumes) must be in this set since it will depend on the size of each disk that you have and how your system is using the space. But, for planning purposes, you need to have about 250 megabytes of space for MPE (FOS). If all subsystems are installed, an additional 295 megabytes are needed. This does not include the amount of space that you must reserve for transient space. Transient space is used for the disk copy of any non-resident object that is not a file (e.g. stacks, system tables, etc.). It is recommended that you have a minimum of 2.5 megabytes of transient space per user. Note that this is a general rule of thumb, and that 'your mileage may vary'. Remember, it is better to err on the high side and avoid the problem of running out of space in the System Volume Set. There are methods you can use to predict this more accurately, but I won't get into that here.

User Volume Sets are optional (obviously). You may have as many as you like. Of course the number of sets that may be mounted simultaneously is limited by the number of disk ldevs that are supported. In terms of overhead, each User Volume Set will require approximately 96 megabytes of space (on the master volume only) for disk resident data structures, most of which is used for MPE/iX's Transaction Manager logfiles. For this reason, it might be a little overkill to define each ldev as its own unique volume set.

BENEFITS OF USER VOLUME SETS

Only the System's Volume set (MPEXL_SYSTEM_VOLUME_SET) must be mounted and available in order for your MPE/iX system to function. As a result, the frequency and the length of INSTALLs can be reduced, resulting in significantly increased recovery time. This is the primary benefit. The following two basic scenarios that may occur will help to illustrate this point.

1. If one disk in the MPEXL_SYSTEM_VOLUME_SET has a hardware problem (such as a head crash), the system will be down while the disk is being repaired, while you are doing an INSTALL, rebuilding the System Volume Set, rebuilding the system directory, and Restore'ing all of the files that belong to the volume set. However, if you have taken advantage of User Volume Sets, you WILL NOT have to rebuild them. Depending on the size of your 'disk farm', this can save an enormous amount of downtime!

2. If one disk in a User Volume Set has a hardware problem, the system should not go down at all. The only users that should be affected are those that are using the volume set. After the disk is repaired, then you only need to rebuild the User Volume Set and Restore all of the files on that set. (If the system does happen to crash due to an disk error in a system critical area of the User Volume Set, you can simply take that disk offline and reboot the system. All other users will then be allowed to do useful work while the disk is being repaired).

There are two other advantages of using User Volume Sets, although these do not have a direct effect on high availability. Depending on how you define your volume sets (and which accounts and groups you place on them), it is easier to manage your disk space usage. It also allows an additional level of data privacy and security.

For detailed information about what commands are required to set up a User Volume Set, please refer to the "Volume Management Reference Manual" (P/N

32650-90045). Also, I would like to refer you an excellent article written by Sandra Iwamoto of Hewlett Packard, entitled "Beyond SYSGEN/VOLUTIL: Volume Management on MPE XL". This article may be found in Volume 1 (paper number 3019) of the Proceedings from the 1990 INTEREX conference that was held in Boston. (I would like to thank Sandra for allowing me to 'borrow' some of the information from her article for inclusion here).

DISK ARRAY TECHNOLOGY

A disk array is a subsystem that consists of multiple disk mechanisms under the command of one controller. A disk array offers several distinguishing features that differentiate it from a traditional multi-controller disk storage system. (For you old timers, HP's 13037 disk controller (7920's, 7925's) was a multi-unit controller, but it did not use disk array technology). Some of the features of a disk array are:

* Data Protection

* Increased Storage Capacity

* Performance Flexibility

There are many different methods that may be used to accomplish these objectives (or features). To understand more about disk arrays in general, you should refer to the 'DISK ARRAYS' primer, offered by Hewlett Packard's Networked Server Division (P/N 5091-1396E). This primer describes the various RAID concepts and RAID levels. RAID is an acronym derived from a paper entitled "A Case for Redundant Arrays of Inexpensive Disks", written by Patterson, Gibson, and Katz from the University of California at Berkeley, published in 1987.

C2252HA and C2254HA ARRAYS OFFER HIGH AVAILABILITY

HP has introduced two models of HP-FL disk arrays (RAID 3 if you care) that offer increased storage capacity, improved speed, AND most importantly, data protection or high availability. The C2252HA and the C2254HA offer storage capacities of 2.7 and 5.4 gigabytes (GB) respectively. The C2252HA comes with three 5.25 inch diameter, 1.36 GB mechanisms; two of which are used for data storage and the third is used as a data protection mechanism. This data protection mechanism is often referred to as the parity disk. The C2254HA comes with five mechanisms; four for data and the fifth for data protection. (For the record, the C2252B and C2254B models offer all of the same features as the C2252HA and C2254HA except that they do not offer the added data protection since they do not come with a parity disk).

The HA models are distinguished by their ability to tolerate an outright failure of any single disk mechanism within the device without losing any previously stored data and without interrupting normal host processing.

OPERATION AND ADDRESSING

The disk array is only supported on MPE/iX in 'striped' mode at this time, and will be treated as a single logical device. Data striping is a technique used by the disk array controller to operate all the disk mechanisms within the device in parallel in a way that boosts the data transfer rate to and from the device. When writing data to the device, the array controller spreads the data across the drive mechanisms in a round-robin fashion, and then reverses the process when reading the data back from the device. In addition, data striping allows a single device (ldev) to have a much greater storage capacity.

Although the following example is not technically precise, it serves well to illustrate the concept of data striping. For example, in an array with two data mechanisms (2-way stripe), the first data byte is written to the first mechanism, the second to the second, the third to the first, fourth to the second and so on. A similar distribution occurs in an array with four data mechanisms (4-way stripe). In either case, the mechanisms inside the array are synchronized so that successive data bytes are read/written from the same location on each mechanism at the same time.

The size of the stripe group (number of data mechanisms in the array) also determines the device block size (logically the same idea as a sector but a new term to avoid confusion over sizes). The mechanisms inside the array have 512 byte physical sectors. Thus, due to the alternation of data bytes between both mechanisms in a 2-way stripe, it takes a total of 1024 data bytes to fill the physical sectors on both data mechanisms which gives a device block size of 1024 bytes (2 x 512). In four-way stripe, it requires 2048 data bytes to fill the physical sectors on all four data mechanisms which gives a device block size of 2048 bytes (4 x 512).

To minimize the impact of this variability on the operating system, the convention is maintained that all levels of software above the disk driver (eagl_disc_dm) will continue to use 256 byte logical sectors with the eagl_disc_dm making the necessary conversions to and from device block addresses. In other words, if you do a :LISTF, you will see the file size reported in terms of 256 byte logical sectors, regardless of the physical sector size or block size of the disk.

In addition to larger capacities, data striping also allows for an increase in raw transfer speed. In the C2252 array (2-way stripe), the burst transfer speed is roughly twice that of a single disk. Likewise, in the C2254 array (4-way stripe), the burst transfer speed is about 4 times that of a single disk! However, it is extremely unlikely that you will see this much improvement in terms of overall system I/O throughput.

On a device to device comparison with a 7937 or a C2204, raw transfer speed is significantly faster. Sequential read performance may be as much as 2-3 times faster in a stand alone environment. Transfers with larger block sizes will also see performance improvements. However, you must be careful to account for the the overall system's demand for disk I/O's per second. Large numbers of older technology disk drives may provide a total of more I/O's per second than one or more disk arrays, even though the total capacity is the same. Again, 'your mileage may vary', depending on your disk I/O requirements.

HIGH AVAILABILITY OPTION

The C2252HA and C2254HA use the parity mechanism to hold data recovery information. As data bytes are written to the data mechanisms of a stripe group, the result of an exclusive OR (XOR) of those bytes is written to the parity mechanism. This is handled transparently by the array controller with no performance impact.

The data recovery information is used to re-create data residing at an unreadable disk address during a sparing operation or if an entire data mechanism becomes inoperable. This allows the array to remain in operation as if no error had occurred. Without the parity mechanism, a hard disk error would be returned resulting in either data unavailability or a system interruption.

NO MORE ERRORS DUE TO A 'BAD SECTOR'!

With a high availability array, 'bad sectors' will be corrected on- the-fly (that's a technical term) and will not cause any interruption to your system's operation. Before examining how the parity mechanism enhances the autosparing process, a brief review of sparing is in order. (Note that all of HP's newer disks, such as the C2204 and 7937, support autosparing. For reference, the 7935 does not).

When a read fails (unrecoverable data) or succeeds with difficulty (marginal data), the eagl_disc_dm issues a spare command.

The disk controller will:

- attempt to save the data (from the 'bad' block)

- spare the block (i.e., remap the block address to a new physical location).

- rewrite the saved data to the block address

If the original data could not be successfully saved, the controller will write its best guess of the data back to the address but with bad ECC (so that reads of that address will fail until the block is rewritten).

After a successful spare, the eagl_disc_dm does another read of the address and will only make an entry in the BST (Bad Sector Table) if the read fails. During the next volume mount, these BST entries will be processed and the user will be informed if the address was part of a permanent file.

SPARING'S USE OF THE PARITY MECHANISM

During the spare, if the original data cannot be read from the address, the XOR information from the parity disk is used to re-create the data before writing it back to the disk. Now, the device manager's read (retry) of the spared address will succeed and no entry will be made in the BST (since no data was lost).

WHAT HAPPENS WHEN A MECHANISM FAILS?

The array controller will remove a mechanism from use when a serious hardware malfunction is detected. If the parity mechanism is not present, the array becomes unavailable. With the parity mechanism installed, the array enters data recovery mode. In this mode, reads and writes to the array succeed at full speed without any degradation in performance. However, the data from the failed mechanism is being created on-the-fly by the array controller using information from the parity mechanism.

The operator is advised of a failed mechanism in a High Availability array through a console message that repeats until a pending reply is answered. Every minute, the following message will appear on the console:

```
DISK ARRAY HAS DISABLED A MECHANISM IN LDEV #.  NOW IN
DATA RECOVERY MODE.  NO DATA LOST OR CORRUPTED.
OPERATION MAY CONTINUE.  PLACE A SERVICE CALL SOON.
```

The console request message is:

```
ACKNOWLEDGE DISABLED MECHANISM IN DISK ARRAY IN LDEV #
(Y/N)?
```

REPLACING A FAILED MECHANISM

The disk array supports "hot" replacement which allows a failed mechanism to be removed and replaced without the need to remove power from the array. This allows the disk array to remain online for certain types of repairs. In a High Availability configuration, after the failed mechanism is replaced, a REBUILD command must be sent to the array through the disk diagnostic (FLEXDIAG) to recreate the new mechanism's content. The array will continue to respond to reads and writes with no performance degradation during the rebuild process. However, the length of time required for the rebuild process to complete is dependent on array activity (from 30 minutes with no array activity to about two hours with very heavy array activity).

As a reminder to the operator (or CE), whenever a new mechanism is inserted into the array, a console message will be displayed and will repeat every minute until a pending reply is answered. The repeating message is:

A DISK MECHANISM HAS BEEN ADDED TO THE DISK ARRAY IN LDEV XX. TO PUT THE ARRAY IN HIGH AVAILABILITY MODE, YOU MUST ISSUE THE REBUILD COMMAND FROM THE DISK DIAGNOSTIC (FLEXDIAG).

The console request message is:

ACKNOWLEDGE NEW DISK MECHANISM AND NEED TO REBUILD IN LDEV XX (Y/N)?

NOTE: As stated earlier, when a mechanism fails, (in a High Availability configuration), the controller will remove the mechanism from the device's logical unit, and a REBUILD is required before it will be added back into the unit. It is important to note that there are two other actions that can cause a mechanism to be removed from the array's logical unit. One is a 'SKIP MECH' command from FLEXDIAG, and the other is the physical act of pulling one of the mechanisms out of the array. The gotcha is that if you've done that, (say to show off how snazzy this thing is), when you put it back in, self test completes just fine and the little green light comes on. It would appear that everything is normal just like before. WRONG! A REBUILD command MUST be issued before the controller will add the mechanism back into the array's logical unit.

FLOOR SPACE SAVINGS

As long as we're talking about the features of the C225X disk arrays, I can't pass up the opportunity to tell you how much floor space you'll save! The C225x array is compatible with HP's new EIA 19-inch rackmount cabinet.

Up to five arrays can be racked in a 19-inch, 1.6-meter high enclosure. This is a total of 27.2 gigabytes of space in one cabinet (about the size of an old Series III)! That's roughly the equivalent of 20 C2204's, 48 7937's, 69 7935's or 232 7925's (for all of us old timers)!!!

C225X DISK ARRAY BENEFITS

In summary, HP's new C225X disk arrays offer increased storage capacity, improved speed, floor space savings, and lower cost of ownership. However, the BIG feature of these arrays is the data protection offered with the high availability option. It is no longer necessary for your application to abort due to a media error in a file or for the system to abort because of a media error in a critical system data structure. These errors will be handled on-the-fly, and you'll never even know they happened unless you look in the system logfiles or in the internal error logs on the drives themselves. In fact, as mentioned earlier, a high availability array can tolerate an outright failure of one of its mechanisms and can be repaired online without interrupting your system's operation.

DISK MIRRORING - OVERVIEW

The C225XHA disk arrays do not totally replace the need or the benefit of the Mirrored Disk/iX product (P/N 30349A). While these high availability disk arrays can tolerate an entire mechanism failure, they are not completely fault tolerant. For example, if the array controller board failed, the array would be unable to operate. In the best case scenario, this would result in one User Volume Set being unavailable to the users until the controller was replaced. However, a failure like this could happen on a system with several disks, ALL of which are in the MPEXL_SYSTEM_VOLUME_SET, and the entire system would be unavailable until the problem was fixed. In a worst case scenario, the problem could be of such a nature that the entire system would have to be re-INSTALLED from the backup tapes that were supposed to have been created the night before (but someone never got around to it)! That system manager obviously had not read this article!

Disk mirroring provides extremely high availability by completely duplicating the entire disk hardware subsystem from the interface card in the HP3000 on out to the disks themselves. High data availability is achieved by automatically maintaining identical information on two partner disks. When an application writes to a disk, the disk mirroring software causes a write to occur to both partners. The application is unaware that this is happening.

Once disk mirroring has been set up (using the VOLUTIL utility), a mirrored disk behaves just like any other disk UNTIL a failure occurs. If either disk of a mirrored pair fails, the system continues to operate normally by accessing the good disk. When the partner has been repaired and is ready to resume operation, the system copies the data from the good disk to the repaired one to bring them back into a consistent state. Once they are in a consistent state, mirroring begins again. All of this happens without interrupting normal system operation.

DISK MIRRORING - SETTING THEM UP

The Mirrored Disk/iX product is available with MPE release A.30.00 (2.0) or later. Prior to release B.40.00 (4.0), only those disks that can be connected using the HP-FL interface are supported. Starting with release B.40.00, disks that can be connected with the SCSI interface will also be supported. Disks in the MPEXL_SYSTEM_VOLUME_SET may not be mirrored. Only User Volume Sets may be mirrored, but as I've stated earlier in the article, you would be wise to use them anyway. Mirrored partners must be the same model of disk and must be connected on separate interface cards. It would also be a good idea to put them on separate channels if possible.

In order to set up a Mirrored Volume Set, you must first initialize it, then add additional members to the set, and lastly, you must create the accounting structure on the volume set and put files on it. The entire process is very similar to setting up a 'normal' User Volume Set.

A Mirrored Volume Set is created by using VOLUTIL's NEWMIRRSET command to initialize the master volume of the set. To add additional members to the Mirrored Set, you will use VOLUTIL's NEWMIRRVOL command. The accounting structure will need to be set up in the exact same manner as you would use for any other User Volume Set. That's how easy it is!

DISK FAILURES

There are two basic flavors of errors with Mirrored Disks that I will attempt to explain. The first situation we will look at is when one of the partner disks does not mount at system startup. The second scenario will describe what happens if a drive becomes disabled during operation.

The system will attempt to mount each member of a Mirrored (or User) Volume Set whenever a new member is added to the set, whenever a drive is powered on, or when the system is booted. It is possible that one or more mirrored partners may be missing or may not be responding. For example, if ldev 32's partner does not mount, ldev 32 is placed in a PENDING state and a repeating console message will be displayed to alert you of this condition. The message will appear every 30 seconds and will look like this:

```
?09:09/12/MIRRORED PARTNER MISSING FOR LDEV# 32
```

The following console request message will also be displayed and you must reply to it to stop the above message from repeating.

```
?09:09/22/ACKNOWLEDGE MIRRORED PARTNER MISSING FOR LDEV#
32(Y/N)?
```

You will not be able to access ldev 32, due to its PENDING state, until you either power on the partner drive (assuming that's why it didn't mount) or until you issue VOLUTIL's SUSPENDMIRRVOL command. This command will allow the drive to be accessed without mirroring. Note that other mirrored drives in the Mirrored Set will continue to be mirrored if both partners are present and accessible. If ldev 32's missing partner is mounted while ldev 32 is still in the PENDING state, mirroring will begin automatically. However, if you decided to issue the SUSPENDMIRRVOL command, once ldev 32's missing partner becomes accessible again, you will need to issue VOLUTIL's REPLACEMIRRVOL command to resume disk mirroring and to start the repair process. Note that the missing partner must be in the SCRATCH or UNKNOWN state before the REPLACEMIRRVOL command can be used. (I'll describe the repair process a little later).

The second scenario occurs when a drive is disabled by the system because it is not responding to I/O or because it is returning errors. For this example, assume that ldev's 32 and 33 are mirrored partners and ldev 33 has had a hardware problem. The system will disable ldev 33 and continue accessing the good partner (ldev 32) without requiring any intervention on your part. The operator will be notified of this condition by the following console message that will repeat every 30 seconds. The repeating message is:

```
?09:09/12/MIRRORED VOLUME DISABLED ON LDEV# 33
```

In order to stop the message from repeating, you must then reply to the following message:

```
?09:09/22/ACKNOWLEDGE MIRRORED PARTNER DISABLED ON LDEV#
33(Y/N)?
```

Once the drive is repaired or replaced, it will mount in the DISABLED state. Again, you must issue the REPLACEMIRRVOL command to resume disk mirroring and to start the repair process.

THE REPAIR PROCESS

The repair process is a Mirrored Disk operation that copies data from the good drive to a partner drive that was previously disabled in order to bring the mirrored pair to a consistent state. This operation is done 'in the background' and does not interfere with normal access of the good drive. After the repair process is finished, normal mirroring is resumed.

The system will automatically start the repair process when mounting the Mirrored Set if it determines that one partner is not consistent with the other. A good disk (source) and a bad disk (destination) are identified and the repair process will start. All data on the source disk is available while the repair is taking place. Additionally, if a system crash had occurred, a repair will be initiated automatically. Lastly, a repair operation will begin when VOLUTIL's REPLACEMIRRVOL command or JOINMIRRSET command is issued. To minimize the impact of the repair process on system performance, only six repair operations may be in progress at the same time. If more than six repairs are required, they are staged and will occur in the order that they mounted.

SPLIT VOLUME BACKUP

Split volume backup allows you to temporarily suspend mirroring in order to backup the data while the system is still accessing the volume set. This is one way to accomplish an online backup, but obviously you will not be protected by disk mirroring during this time. There are three basic steps to doing a split volume backup. You must split the Mirrored Set, perform the store, and join the Mirrored Set again.

To split the set, all access to the volume set must be terminated. You can do this by having the accessors logoff or simply change groups to a group which does not reside on the volume set. At this point, you need to do the VSCLOSE {setname}; SPLIT command. This will complete successfully only if there are NO files being accessed on the volume set. All members of the volume set will now be in the LONER state. Now, you should issue the VSOPEN command to make the volume set available again. You will notice that one of the partner drives will mount as a 'SPLIT USER' volume and the other will mount as a 'SPLIT BACKUP' volume. Next, the STORE command with the SPLITVS parameter should be issued. When the STORE is complete, use VOLUTIL's JOINMIRRSET command to join the partners once again and to start a repair process.

CONCLUSIONS

As you can see, the system manager has three different options to choose from in order to configure a disk subsystem for high availability. Any one of these options may be used independently or a combination of them may be used.

I highly recommend that every system manager of an HP3000 system installation consider the use of User

Volume Sets. They are free! They cost you nothing except the time required to plan for them and set them up. In the event of an untimely disk failure (and they always are), you will significantly decrease your recovery time. You will be glad you made this decision!

Secondly, the new disk array technology is exciting and will offer tremendous benefits in terms of higher capacities, lower cost per megabyte, floor space savings, but most of all, high reliability; especially when the parity disk is being used. As stated earlier, the C225xHA arrays will allow the system to continue normal processing during a large percentage of disk hardware problems. In fact, they can tolerate an outright failure of one of the mechanisms within the device without losing any data or causing any interruption to your system's operation. And, in most cases, they can even be repaired online! Additionally, since disk mirroring can only be done on User Volume Sets, this option allows a system manager to get added data protection for the System Volume Set.

Lastly, Mirrored Disk/iX provides a high availability solution by using redundant disk hardware. Mirrored disks are a great solution for systems that need high availability. It is a more expensive choice since you must invest in a redundant disk hardware subsystem, purchase the Mirrored Disk/iX product, and you must be willing to put your user's data on one or more User Volume Sets. However, it also offers the most protection from costly downtime.

If your shop is one that simply cannot afford ANY downtime, you may even wish to consider the idea of mirroring the new disk arrays. This option which combines all three methods, provides the most protection against downtime due to disk failures. The C225xHA arrays will provide protection against the majority of disk failures during the times you would be vulnerable in a Mirrored Disk environment. As an example, you are vulnerable in a Mirrored Disk environment when a split volume backup is being performed or if a hardware failure (such as an interface card) occurs, causing one of the partner drives to be disabled. The choice is yours. As the man in the commercial once said...you can pay me now or pay me later!

MPE/iX High End Kernel Algorithms for OLTP Performance

408-447-6273

Authored by: Thomas McNeal
Presented by: Hung Nguyen
Commercial Systems Division
Hewlett-Packard Co.

The MPE/iX Kernel and Memory Manager support high end, high performance OLTP applications through a tightly coupled interface with the MPE/iX Transaction Manager, as well as the MPE/iX File System and IO System. The Kernel manages a 64 bit virtual address space mapped into large physical memory - on the order of gigabytes - through intelligent predictions of data access patterns, fast response to physical page requests, fast discarding of pages which will not be accessed in the near future, highly parallel output request pathways, high IO request bandwidth, and minimal synchronization with data residing both in main memory and on secondary storage.

The terms 'high end' and 'high performance' must first be defined in order to determine the rationale for different Memory Manager features, and what meaning they have to the user. These terms indicates a system using hundreds to thousands of megabytes of main memory, connected to tens to hundreds of secondary storage devices, or spindles. Thousands of users and/or processes are supported in a mixed - batch and interactive - environment. The system performs hundreds of transactions per second, with uniformly low response time. In most benchmarks, response time must be below two seconds for at least 98% of the transactions, but the goal for low response time is generally considered to be subsecond response time.

The problems which must be addressed when providing this scale of performance and capability are numerous, varied, and challenging. They include continual monitoring and synchronizing global resource tables, managing high IO request frequencies, maintaining a high IO bandwidth, reducing overall transaction pathways and removing transaction bottlenecks. The response to these problems may be characterized into two broad issues: maintaining high overall resource capacity and low execution delay frequency.

Specific Intent of Memory Manager Structure and Design

To solve these problems, we must consider the responsibilities of the Memory

Manager. It provides one essential resource to the rest of the system - pages of main memory. Ideally, it should provide this resource at infinite speed, and with no perturbation to the rest of the system. Realistically, of course, it will make demands upon the system for its own resources, and will both propagate delays from system resource requests, and create delays of its own when fulfilling the requests made of it.

The real challenge in enhancing OLTP performance is to reduce the effect of these delays by anticipating system needs, and preparing resources in advance. To fulfill demands most effectively, and to provide a scalable architecture which can support a wide range of system configurations, the Memory Manager supports a "demand paging" environment, together with extensive "prefetching" - providing pages of main memory upon demand, and, based on intelligent predictions, bringing in pages likely to be needed soon.

The Memory Manager places few restriction upon requests, but allows the process execution priority to select which process is able to make system demands. It attempts to prepare for all demands by maintaining global resources designed to meet these needs, and relies heavily upon information from other subsystems to predict future system behavior in order to prepare resources before they are needed. This is particularly critical when attempting to avoid process execution delays harmful to transaction performance.

The MPE/iX Memory Manager has adopted the philosophy that reducing its own demand for critical resources is as necessary for high performance as is improving other subsystems response to resource requests. Noting that IO performance is a prime component of system performance, and with the observed trend of processor speed outstripping IO speed, the best way to improve IO performance from the Memory Manager's point of view is to avoid IO dependencies. This is possible by reducing the need for IO and, as a consequence, reducing the number of IO interrupts.

When IO is required, the Memory Manager attempts to make it as efficient, and with as high a bandwidth as possible. The cost of each IO must be low, and serial dependencies must be removed or minimized. In particular, blocking on IO requests must be avoided whenever possible.

Finally, the Memory Manager, as well as the rest of the MPE/iX Kernel, uses its detailed knowledge of critical database application designs to minimize the impact of user requests, and to maximize system performance.

Global Clock Protocol

To make page usage decisions, the memory manager uses a 'global clock' page replacement protocol, similar to the protocol used for IBM's VM/SP page replacement. A global clock protocol treats main memory as a single pool of pages, making page frames throughout memory available to any process in the system at any time. This is in contrast to many other implementations, such as UNIX, and DEC's VMS, which assign a number of physical page frames to each process, and to BIOS, and MPE-V, which both define a limited system data area for file buffer management.

In the former example, each process is limited to using a small, exclusive pool of page frames, regardless of the activity of the system as a whole. Thus, when a process has opened too many files, or created too large a stack, it could start competing with itself for page frames. This kind of behavior, where pages needed by a process are thrown out of main memory to make room for other, equally necessary pages, is called 'thrashing', and virtually guarantees poor performance. When a limited system data area is used, as in the latter example, the operating system reserves a pool of page frames for its own use in managing file system and cache buffers. The use of the global clock protocol, together with the use of virtual memory mapping into global physical memory, avoids these limitations by treating main memory as a single resource for all virtual mapping. No separate process pool, file system buffer or cache buffer area exists, and mapping can occur anywhere in physical memory.

The clock protocol fills a global table, known as the Recoverable Overlay Candidate (ROC) pool, which provides physical page frames for all processes. The clock protocol treats each physical page in main memory as a single tick on a clock face, with a clock "hand" circularly traversing main memory to determine if a page has been touched since the clock hand last visited it. If not, and if it is otherwise available, the page will be added to the ROC pool.

Each page in the pool is left mapped in virtual memory, remaining instantly accessible to all processes in the system. If it is referenced while still in the pool, it will be "recovered", or yanked out of the pool, to prevent other processes from later replacing it in memory.

As each page is added to the pool, it is written out to disk, if the copy of the page in secondary storage differs from the copy in main memory. To make sure that enough time will elapse before a page recently added to the pool is actually used, and to therefore increase the likelihood that any write in progress for a page in the ROC pool will have been completed before being used as a replacement candidate, the pool size scales with the size of main memory, and is carefully tracked to maintain a minimum number of available pages.

This is particularly important for performance. If replacement candidates have outstanding write requests at the time they are taken from the ROC pool for incoming pages, the read for the incoming page must be delayed until the outstanding write completes. This increases the page replacement latency time, with a corresponding increase system overhead required to manage the competing IO requests. With a global pool, all processes we can be assured of a supply of available page frames for read requests, without having to force the read request to wait for a write to synchronize memory with secondary storage, and without depending upon limitations imposed by a using a fixed number of page frames per process.

In contrast to the VM/SP protocol, the clock is not always running, but is invoked when the ROC pool runs low - that is, when physical page frames which can be used for incoming pages grow scarce. This technique has proved to be both efficient and highly scalable. In particular, the original global clock was designed for systems using tens of megabytes of pages (centering around 32-64 megabytes), but has remained unchanged with good performance in gigabyte sized memory.

Process Locality Management

Another process oriented technique - used primarily by IBM's MVS - maintains a process 'working set', a collection of pages related to a particular process which are brought into memory as a single unit before the process is launched. This avoids the unpleasant experience of both excessive and serialized process blocks due to page faults, but experience with the MPE/iX process access patterns suggests that this localized memory handling technique is wasteful. MPE/iX uses a 'locality list', in which the pages required by a process are added to each process's locality list as they are needed - in other words, as they are encountered through page faults. As with the working set technique, pages belonging to a process locality list must be fetched into memory before the process can be launched, but the contents of a process locality list is determined by specific process behavior at the individual page level, and can be modified at the system level by events in the system, and in other processes as well.

In addition to pages on the locality list, stack pages may be brought into memory prior to a process launch, or a page may be "prefetched" into memory as part of a page fault. In the latter case, when the Memory Manager sees page faults in some frequently used objects, such as the Compatibility Mode SL, it will bring in some adjacent pages as well as the faulted page. The term "prefetch" is used to indicate that the Memory Manager makes an educated guess, and tries to bring in pages before they are needed.

In addition to faulted pages, if the memory manager removes a frequently referenced page from main memory to make room for some other page - which is, we hope, more urgently needed - the Memory Manager will place the "kicked out" page on the locality list of the process which last touched the page. This is known as the kick-out locality, and the page will be brought back into memory, along with all other pages in the locality list, before the process next executes. This is another case of an educated guess - the Memory Manager is assuming that a frequently referenced page will be needed by the process which made the last reference, and attempts to avoid any unnecessary process blocks by bringing it into memory before the next process launch.

In most cases, a page on the locality list must actually arrive in main memory before the process is launched, but prefetched pages need not actually arrive beforehand. As the name suggests, these pages will hopefully arrive in memory before they are needed, but since the need is not absolutely established, the process launch is allowed as soon as the request for prefetched pages is issued.

Predicting File Access Patterns

Components necessary for each transaction must be available for use as the transaction needs them to guarantee the highest possible transaction throughput. If the process driving the transaction blocks, waiting for resources, the system overhead for that transaction increases, and other transactions dependent upon the resources used by the blocked process may be delayed.

To prevent such process blocking, the Memory Manager relies on the Transaction Manager and file system predictions to sharpen the focus for memory requests. If access patterns can be accurately predicted, each IO request can minimize the need for future requests by incorporating pages which are likely to be touched in the near future.

For instance, a file's access method can be used to predict which pages may be needed when a process has opened the file. When it first sees a serial access method, the file system will ask the Memory Manager to prefetch eight pages following the required page. The intent is both to insure that pages will already be in main memory when they are needed, and to fold a series of potential requests for adjacent pages into a single request.

In addition, the file system keeps track of the actual file access patterns, increasing or decreasing the number of prefetched pages in accordance with the results of each file access. If the prefetched pages are not yet in memory, the number of pages will be incrementally increased up to 15. Of course, following the global viewpoint which the MPE/iX Kernel prefers, the Memory Manager will not allow file system predictions to dominate when it has detected pressure for memory. If the file system sees that a prefetched page has arrived in memory, only to be kicked out before the page was actually referenced, or if the Memory Manager detects on its own that it is having difficulty finding page frames for replacement, the file system will scale back its prefetch request size, thus reducing the overall workload.

Access patterns are tracked even when the access method is random. When file accesses tend to follow a serial pattern - in this instance, after four consistently incrementing accesses - the file system will start constructing IO requests as if the file access method were serial. If the access patterns later revert to a random pattern, the serial IO requests will be abandoned immediately.

To help with maintaining enough candidates for page frame replacement, and to prevent memory pressure from limiting requests for incoming pages, the Memory Manager allows its customers to indicate when pages can be discarded. When closing files, or when posting the Transaction Management Log to disk, write requests can include a "Make Overlay Candidate" option. This option adds the page to the Memory Manager's ROC list when the write is complete. Since it is very likely that the virtual addresses mapped to these pages will not be touched for some time, the Memory Manager can use these pages for mapping other virtual addresses, at little cost to the system.

In this way, both static and dynamic information is used to streamline the demands made upon the Memory Manager, and global system behavior helps insure the highest resource availability possible.

Reducing IO Request Frequency

The Memory Manager can avoid unnecessary IO altogether when it can determine the contents of a page without requesting those contents from secondary storage. When an object is first allocated - when a new file is opened, or when a new process's data structures are created, for example - the pages mapped to these locations are called "virgin" pages, and the memory manager can fill these pages with the initialization byte specified by the object definition without having to read anything from secondary storage.

Another part of the Kernel, the Virtual Space Manager (VSM), will tell the Memory Manager when it can use virgin pages through a mapping scheme which tells it when pages are either part of a new object, or have not ever been written out to secondary storage. Some events, however, such as file truncation (an 'eof cutback'), are candidates for later usage of virgin pages, but cannot be tracked by VSM, since this type of information is only known to the file system.

Consequently, the file system is also designed to recognize when to use virgin pages. Using the above example, it will send "Virgin Page Prefetch" requests to the Memory Manager when expanding the file after an eof cutback. In this case, VSM has lost track of the fact that the page can be considered virgin, since that page has been written out to secondary storage at some time in the past. Since the file has later been truncated and then expanded, however, the page need not be read in from the disk, but can be initialized as a virgin page.

Reducing IO Dependencies

To reduce dependencies upon IO performance, the Kernel's first responsibility is to keep heavily used pages in main memory, and to avoid redundant reads and writes. Keeping pages in main memory avoids IO altogether, and allows the system resource provided by the memory manager to be available at the time of use.

Redundant writes are easier to control, since the page being written is still part of main memory. When several write requests are seen for the same page, the memory manager will note the current state of the first write, and avoid all others until the write is actually in progress. At that point, there is no guarantee that the page's state in secondary storage actually reflects that of main memory, and the write must be reissued to bring the page's copies in accord with one another.

Redundant reads are actually uncontrollable by the memory manager, since the applications supported on the system drive the IO requests. As an example, knowledge of several important database applications show us that common database designs - based upon the structure of UNIX, and are necessarily built for portability - inherently cause redundant IO in MPE/iX. Since MPE/iX cannot modify the application requests, it must provide the most efficient interface possible for these critical applications.

As a solution for this particular problem, current releases have implemented "Pseudo mapped I/O". This IO uses a form of aliasing that causes data from one virtual address space (a file, for example) to be read, or mapped in from disc into a different virtual address space (a buffer, for example) in physical memory. It combines features of mapped files and raw disc io, and is geared toward applications that do not use MPE XL's mapped file features, but instead use a buffer mechanism.

These applications today copy data from their mapped files to a buffer, process it in the buffer and later copy it back to the (mapped) file which is then written out to the disc. For random access files, handling buffers in this way defeats the underlying mapped file mechanisms in the operating system, and introduces significant overhead in the form of copying data (including cache misses), excessive cache flushes, context-switches, page faults and replacements, and excessive path length.

Using pseudo mapped I/O, the data from the file's page on disc is read directly into the buffer's page frame in memory. When it is to be written out, the data from the buffer's page frame will be copied directly to the appropriate location on disc. Further, the application can request that when the write has completed another page of data be read directly into the buffer, without returning to the caller in between. With Pseudo-mapped I/O, copying data from the file to the buffer and vice-versa is therefore eliminated. This is especially significant considering that a cache miss will be taken on every cache line when copying from the newly mapped in file page, and, when the buffer is about to be copied back to the file, the file page may not be present in memory, causing an additional I/O from disc that was really unnecessary.

Making IO Efficient

A critical aspect of removing redundant IO is reducing or eliminating the synchronization with secondary storage required by older systems, such as UNIX, which do not have a integrated, log-based Transaction Manager. The durability required by transactions forces these systems to constantly synchronize main memory with secondary storage. In MPE/iX, however, the Transaction Management log allows implementation of the "write ahead" protocol by capturing the transaction image in the Transaction Management log, and allowing the IO which actually synchronizes the modification with the disk to float out to secondary storage whenever the Memory Manager decides is most convenient.

The only restrictions on this capability is that the contents of the log itself must be written to secondary storage first, before any files or other objects modified in the transaction are written out (revealing the meaning of the phrase "write ahead"). Using this guarantee, we can recover completed transactions from the Transaction Management log after a system crash, since the log contains a copy of data structures involved in the transaction, taken both before and after the transaction modifications. The Transaction Manager can then use the log and the old value of the data structures to apply the modification as part of the crash recovery process.

The freedom to abandon constant synchronizations allows the Memory Manager to bundle many modified pages together in a single IO, reducing IO frequency and increasing IO bandwidth for transactions. This bundling is called a 'gather write', which occurs when the Memory Manager is performing its own page replacement resource housekeeping. When doing this housekeeping, the memory manager must write pages in order to reserve them as candidates for replacement by pages needed from the disk. When building the IO request, the memory manager will gather in any adjacent pages (adjacent in virtual space, that is) which are also dirty and may need to be written out. The pages included as gather write pages do not increase the cost of the IO, which must be performed for one page anyway, and may avoid subsequent IO for the extra pages included.

Removing IO Serialization

Gather writes generally take place with IO generated by the memory manager itself, as opposed to user requested IO. For the latter, it is often necessary - particularly with IO requests from the Transaction Manager - to wait for the completion of each IO request before continuing. This can be particularly damaging to throughput, for when forced to wait upon each IO completion before sending the next IO request, transaction performance becomes dependent upon IO performance. If the IO requests are serialized, transaction completion will be slowed to the speed of the IO devices.

For this reason, the Memory Manager and Virtual Space Manager can now handle parallel IO requests - requests for up to 32 IOs which can span several virtual address ranges, and several IO devices. This capability - know as 'Post Ranges' - allows requests for several IOs at once, and will break down each IO request into several more requests, depending on whether the requested ranges spans more than one IO device. By sending requests to many devices at once, IO work can then continue in parallel, and the Memory Manager can return IO completion messages back to the requestor, either as a bundle, with one message for all requests, or asynchronously, with a separate message for each IO range. The Transaction Manager uses this capability both during the checkpoint process, when it is flushing its logs out to secondary storage, and when recovering data from the logs during startup time.

This capability has been particularly useful for both proprietary and third party database products. With IO serialization removed, checkpoint time has been reduced significantly, with the consequential increase in OLTP performance due to the removal of the checkpoint process dependencies.

Summary and Conclusion

The MPE/iX Memory Manager uses a number of sophisticated techniques to increase OLTP performance by reducing the dependencies which transactions encounter when making requests for resources, and by increasing the availability of resources. As part of the MPE/iX Kernel, it maintains a global view of resources, while allowing individual processes to control their own usage and release of resources. It allows the file system to use access pattern predictions to prefetch memory before it is needed, permits Kernel users to release resources after use, and employs the gather write protocol to include all possible IO in a single write request.

Together with the Virtual Space Manager, the Memory Manager avoids IO overhead for virgin pages, divides IO requests into parallel groups, and tries to maintain efficient IO access through virtual mapping and pseudo-mapped IO. With the cooperation of the Transaction Manager and File System, it avoids IO overhead on truncated files, reduces checkpoint overhead with grouped IO requests, and implements the write ahead protocol to reduce transaction synchronization dependencies. With these features, the MPE/iX Kernel sustains increasingly high OLTP work loads, and maintains increasingly high performance.

There are, however, many challenges yet to be solved. The higher capacity systems will be called upon to support more memory, and more secondary storage, than ever before. Hardware reliability will be a key issue, particularly when considering the sheer number of pages in main memory, and the amount of information which must be kept consistent with secondary storage. Software reliability and capacity is a key item as well, requiring both new and evolving designs to support OLTP performance, and to guarantee system reliability, transaction speed and durability, and user responsiveness.

In addition, IO performance is a key component when considering Kernel algorithms, so we must continue to be concerned about the widening gap between processor speeds and IO speeds, as well as the increasing demand for secondary storage memory, or virtual space, in the context of MPE/iX. These demands raise the issues of IO bandwidth, overall reliability, and overall cost, which are being addressed by many differing approaches throughout the industry, including disk array (RAID) technology to increase bandwidth and reduce costs, hardware and/or software data striping techniques to increase parallel data transfer operations, linear data organization on the disk itself to reduce device latencies, and other many other techniques.

MPE/iX continues to attack these problems to provide efficient, responsive, and state of the art OLTP performance support for HP's commercial operating system products. As part of the leading edge commercial technologies, MPE/iX is committed to maintaining its leadership role in providing the highest reliability and highest performance OLTP support.

References:

The Evolution of Paging Algorithms in VM and MVS
Thomas Beretvas, IBM Data Systems Division

On-Line Transaction Processing Performance Comparison of MPE-XL and HP-UX
Gary Grossman, Hewlett-Packard Co.

Strengths and Weaknesses of the MPE/XL Operating System
Jeff Spirn, Hewlett-Packard Co.

Paper 3805

TurboIMAGE Applications in the '90s and Beyond

RJ Diepeveen
Hewlett-Packard Co.
Commercial Systems Division
Cupertino, California

If your site uses TurboIMAGE for data management, how do you, as manager or programmer, make the decision to implement new or revised applications on a new database technology? What application requirements are so stringent that your current storage management system is rendered obsolete? How is the retraining of personnel and data conversion justified in the current economic environment? The intent here is not to specifically answer these questions, but to describe ways of taking your existing TurboIMAGE based applications and enhancing them in changing business environments.

If your site is typical of many in the HP 3000 user community, you are probably using COBOL to manipulate VPLUS screens and TurboIMAGE to store your information. Batch processing is, no doubt, a part of the daily or weekly schedules. Interfacing with your HP 3000 is more than likely done with either a plain old terminal or via personal computer and perhaps a LAN. In many ways, the HP 3000 application environment has changed very little over the last decade. But new and exciting things are happening with database technologies, user interfaces, and application development, not to mention the incredible advances with processors.

Today, most non-relational database applications are compared with relational database models. We will focus on TurboIMAGE and the relational model. The software implementation of the relational model provides logical transaction integrity, dynamic and flexible database design, dynamic retrieval methods, standards of access and client/server computing capability. Nearly every relational database vendor works to have these attributes in their product. What do you do if you want these features in your current TurboIMAGE based application? The following paragraphs describe these concepts in greater detail and offer insights into how TurboIMAGE can be used to achieve this functionality.

Logical transactions. What are they? One or more additions, deletions, or modifications of data which represent a complete element of work. In TurboIMAGE, this would be combinations of DBPUTs, DBDELETEs, and DBUPDATEs which collectively represent a whole transaction. Implicit is the reading and tracking (DBGET and DBFIND) of data required to make the database changes.

Previously, logical transactions in TurboIMAGE were static. Static transactions are known to programs which execute them, and recorded in MPE User Logging Files. They are static in that the commitment of these transactions are immediate and rollback of the transactions requires exclusive access of the database by DBRECOV to bring the database to a logically consistent state. Now, with MPE/XL 3.0 (TurboIMAGE A.03.00), dynamic transactions and their recovery are possible. Dynamic transactions are commited or rolled back immediately. Using new intrinsics (DBXBEGIN, DBXEND, DBXUNDO), coding can demarcate dynamic transactions such that they can rollback programmatically, in the event of program abort, or following a system abort during startup. Gone are the days when program aborts or system aborts leave logical transactions incomplete.

Another feature of relational database implementations is that of dynamic and flexible database design. What this implies is that if table (set) definitions require changes such as adding or deleting columns (items), this change can be completed without affecting current applications. How is this different from using DBChange Plus or other database maintenance tool to add or delete items from a dataset? To answer this we must pursue the concept of binding.

Using the aforementioned 'typical' HP 3000 shop for an example, COBOL programs have a working storage section where record layouts and other variables are described. With TurboIMAGE, most programs include buffer layouts for each dataset that is accessed, each broken down into specific data items. For the program to have to know so much about the structure of the database eliminates a great deal of flexibility. Any change to the database requires a corresponding change to the program. This technique is called "early binding."

The other side of the coin is "late binding." Suppose for a moment that the program does not even know the name of the database, let alone any of the structural details. The one thing the program does assume is that the database is a TurboIMAGE database. Three things need to be supplied to the program as input: base name, password, open mode. The program can now, for the password supplied, interrogate the database for structural details, including sets, items, and paths with the DBINFO intrinsic. The working storage section of the program contains indexed arrays for storing and manipulating the structure and data.

Now we have the flexibility to change the structure of the database without having to change the programs which access it. An excellent example of late binding is QUERY.

With late binding we can change the database as business needs dictate. Dynamic changes are not currently possible with TurboIMAGE, as all maintenance tools require exclusive access to the database. Relational databases, on the other hand, allow for dynamic creation of indexes and view tables, and the ability to add and delete columns. Although this functionality appears highly desirable, actual relational based applications rarely use these features. Sadly, late binding is rarely used in any applications because coding for late binding is much more complex and therefore more time consuming. Regardless of the database type, the long term benefits of late binding is well worth the effort required.

Relational databases offer indexes to provide fast access to rows. Hashing, the TurboIMAGE access method, is now being seen in some relational databases as an additional means of accessing data. TurboIMAGE is no longer limited to only hashing. Beginning with MPE/iX version 4.0 (TurboIMAGE A.04.00), indexing is introduced in TurboIMAGE through an integrated interface to sophisticated indexing products developed and marketed by two independent software suppliers. This does mean that an additional product must be purchased to achieve this functionality. However, just as with late binding, the benefits of faster and more complex retrievals exceeds the cost.

Let's examine the retrieval methods now available in TurboIMAGE. True hashing is done with ASCII data stored in the X and U data types and with values stored in types P and Z. Numeric data types including I, J, K, R and E benefit from algorithmic placement within the dataset. In either case, key items are linked to search items along paths. Manual master sets offer referential integrity to related detail data sets. Both Automatic and Manual master sets offer keyed access to detail data sets. Master set access can now increase due to another enhancement to MPE/iX 4.0 TurboIMAGE (A.04.00); CIUPDATE, which allows for the updating of Critical Items, Search and Sort items, found in detail data sets. Items whose values change frequently can now be search items with chained access.

Indexing of data provides access in two basic ways, generic or partial key searches and keyword searches. Generic searches involve supplying leading characters as the basis for the search, with returned entries those that match the search criteria in a sorted order. Keyword searches involve supplying a word which is then found anywhere in an item or collection of items. This indexing is independent of traditional TurboIMAGE keyed access.

The capabilities of these indexing products extends far beyond that discussed here. Suffice it to say, access methods now available in TurboIMAGE rival those of any relational database implementation. More detail on these indexing products can be obtained from the TurboIMAGE reference manual (HP part number 30391-90001, fourth edition) and from the software suppliers indicated therein.

Lastly, as far as access methods are concerned, we must not forget serial access. Albeit slow, this access method is often times overlooked and most certainly underrated.

Referring back to our original list of relational model capabilities, we are left with a discussion of standards and client/server computing. A discussion of standards could be lengthy, so this section will be abbreviated. It is a prerequisite for the client/server computing concepts, as will be seen.

TurboIMAGE is not today a standard, unless, of course, you consider that it is one of the most widely used database management systems. Standards, however, imply that there is some sort of universal acceptance within some context. SQL, or Structured Query Language, has achieved this acceptance. The SQL Select command returns information in the same manner on different database implementations which meet the 'standard.' So how does TurboIMAGE fit into a world of standards? TurboIMAGE understands the SQL language standard.

Database Management Systems manage data. Data is added, deleted, and updated. What the data looks like is of little concern for the database; managing the data is its only concern. Programmatically speaking, to add data with SQL involves the INSERT command. With TurboIMAGE this is done with DBPUT. The end result is the same; data is added! If the program commands INSERT and this is translated to a DBPUT, then SQL is communicating with TurboIMAGE.

At present, some of the SQL functionality for TurboIMAGE can be found with Allbase/Turbo Connect. SELECT commands are converted to DBGET calls and results are handed back to the calling program which operates as though it is working with a database which meets SQL standards. Investigation is under way to determine the feasibility of incorporating additional SQL functionality into Allbase/Turbo Connect. TurboIMAGE could enjoy a multi-lingual existence. In a strange twist, TurboIMAGE intrinsics could be thought of as "Native Mode," making SQL "Compatibility Mode."

So now we have TurboIMAGE talking the SQL language. Just what good is that, anyway? It means that any software written to interface to a relational database which employs the SQL standard can execute against a TurboIMAGE database. It means that software suppliers can develop applications regardless of the database management system chosen by the customer. It means that programmers trained in SQL can use high performance TurboIMAGE databases. It also means that TurboIMAGE and relational databases have more in common than they are given credit for in the press.

Having TurboIMAGE talk SQL also has an additional benefit. Application Development Tools! Numerous software suppliers offer products which talk SQL and which can now talk to TurboIMAGE using Allbase/Turbo Connect. This leads, in part, to the last of our original topics: client/server computing. Several vendors offer client/server SQL application development tools which can now work with TurboIMAGE.

Years ago, Hewlett-Packard marketed the HP 3000 as the distributed data processing system. At that time, when your business needed more computing power, additional processors were added to a network of minicomputers. Today personal computers and workstations sit on desks instead of dumb terminals, and the concept of distributed data processing means that the work is done quite literally at your fingertips. Where before the networked HP 3000s shared the load, now they act as servers to the multitude of processors requesting data.

Already now, with Allbase/Turbo Connect, client/server computing can be realized with TurboIMAGE as the database of choice. But that is not the only way to achieve the client/server relationship. Using some of the late binding concepts discussed in this article, and taking advantage of the varied retrieval methods, it is possible to develop your own TurboIMAGE server for your client processors to communicate with. TurboIMAGE based applications today within Hewlett-Packard enjoy both a direct, program to database interface and a client/server interface. It is simply a matter of creating server software which works with TurboIMAGE and communicates with client applications. This software can be database specific, or generalized for any TurboIMAGE database. But the point is, client/server computing with TurboIMAGE can be done. This description is a bit vague here, but we will return to client/server computing in an upcoming example.

To again list the original five relational areas, we have logical transactions, dynamic database design, dynamic retrieval methods, standards, and client/server computing. At this point, a specific example might help clarify some points and give some additional ideas for your own projects.

Let's use an HP 3000 based Inventory Control/MRP application as the basis for this example. The database consists of a part master, inventory by location, material purchasing, vendor information, bills of material, routings and work in process. This list is not all inclusive, but will give us a good start. Let's also suppose that the product design department has installed CAD/CAM systems on an HP 9000 platform and quality control has HP Vectra PC based statistical software. Users of the system include production scheduling, manufacturing, design, material handling, shipping, quality control and accounting.

Incoming inventory receipts will update quantities in stock locations, but each delivery also updates purchasing and vendor information. This constitutes a logical transaction and will be handled by TurboIMAGE as a dynamic transaction. The same is true for workorder pulls to work in process. Many parts are affected and the logical transaction must again be protected. As a final example of logical transactions, the design department develops bills of material and routings for final product and component parts. This work is being done on the HP9000 CAD system and the data is uploaded to the HP 3000. A lot of TurboIMAGE datasets are affected by each of these uploads. Again, dynamic transaction protection is essential.

The cost accountants have requested that additional information be collected for each component part in the part master where lower level labor and material cost apply. Quality control has requested that additional statistical information be collected in test phases associated with certain stages of electrical component part production. Production scheduling wants more detail in routings to accommodate sub-contract assemblies. The design department would like to have new items added to the part master, bills of material and routings files so CAD drawings can be tracked. Each of these requests will require database structure changes; new items added to existing datasets and new paths established to automatic masters to improve the interactive reporting needs of each user. If the programs were written with pre-defined database structure explicitly documented, then they will have to all be changed to handle the new items. But if the late binding approach to programming was used, significant maintenance gains will be realized in accommodation of these requests.

Purchasing has the need to see all vendors which can supply a particular part. Quality control would like to inquire against test result fields for ranges of results. Master scheduling wants to see all component parts which contain certain purchased parts and materials. For each request, timely, interactive inquiries are specified, not reports. Some of these requests may be fulfilled by TurboIMAGE hashing, but others may not be possible without serially reading the datasets. Here is a situation where indexing is more appropriate. Indexing the test result items and part numbers will offer fast range and keyword retrievals, fast enough to address the interactive needs of these users.

The marketing department has recently installed a new market analysis package that uses a relational database management system. Input for this application comes from the software supplier as a service, order information stored in the TurboIMAGE database, and sales projections entered by sales representatives via portable computers linked over modem. Allbase/Turbo Connect is used to link the TurboIMAGE database to the relational database and SQL is used to access both. The sales representatives are using client/server computing to manipulate the data at the portable computer while interacting with the server host. This client/server computing is noticed by the quality control department who now requests that test results be automatically passed to their Vectra based statistical package.

To address this request, server software is specifically created to monitor test routings. When test results are reported to TurboIMAGE, the server obtains the data and makes the results available to the client which has a process awaiting this notification. Analysis is done by the Vectra based package and the results are graphically depicted in a window on the monitor. Should the results fall below acceptable limits, the PC, using a modem connection, pages a quality control engineer to address the situation.

The design department, seeing the client/server successes in the other departments, request that this technology be added to their environment. You may recall, the design department requested additional items be added to routings, part master, and bills of material datasets to track CAD drawings. These graphic drawings reside on the HP9000 file server and are managed solely by the CAD/CAM software. It is important that when a part is being engineered or changed the bill of material and routing for this part be frozen and the part master reflect change order versions.

The resulting system allows a design engineer to 'check out' a part, thereby freezing it. As a product or component part is designed or re-designed on the CAD client, the server is protecting source information and assisting in the management of the process. TurboIMAGE stores not only the part data, but drawing references for use by the CAD system including presentation information and procedure names for presentation. Changes to designs, bills of material, routings, and versioning, done by the CAD system, are passed to the server for immediate updating in the TurboIMAGE database. Having TurboIMAGE store not only drawing references, but procedure names as well, hints strongly of an object orientation. Object database methodology is not totally lost with TurboIMAGE.

These are just examples of what is possible, and many other great ideas can apply in this scenario. These examples were selected more for their simplicity of description than as any sort of definitive description of what is possible. The goal here is to demonstrate that the TurboIMAGE database is limited more by our own ingenuity and enthusiasm than it is by technology.

As you can see from these concepts and example, a good deal of functionality typically associated only with more recently released database management systems can be found in TurboIMAGE as it is continually enhancemented. In fact, referential integrity has been standard with TurboIMAGE from the very beginning, whereas relational databases seldom include this in their feature lists.

The basics, however, are covered. Logical transaction declaration and rollback recovery, flexible database designs and retrieval methods, compliance with industry standards, client/server capability; all of these are areas where TurboIMAGE has risen to the challenge. As technology continues to march forward from here, TurboIMAGE will march forward as well. Today, nearly every HP 3000 uses TurboIMAGE databases and with so many businesses relying on it, HP is committed to supporting and enhancing TurboIMAGE in the '90s and beyond.

Acknowledgements

I would like to thank Alfredo Rego for his thoughts on binding and industry standards, Eric Savage and Terry O'Brien for their quidance with indexing, Wirt Atmar for his thoughts on standards and compliance, and Jim Sartain for his committment to TurboIMAGE.

PAPER #3806

Key Buzzwords of the 1990's: Open, Interoperability, POSIX and DCE

Joseph Feiner

Hewlett-Packard Co.
19447 Pruneridge Avenue
Cupertino, CA 95014

ABSTRACT

This paper considers "OPEN", a key buzzword of the 1990's. The paper explains what customers want when they ask for an OPEN system and some technologies implemented on the HP3000 to support this claim. Technologies considered include POSIX and DCE.

The paper goes on to mention that even if the customer doesn't intend to use OPEN features in the short-term, that these developments will provide important long-term benefit. As a result of the availability of these new services, many new HP3000 applications will become available; applications will be easier to port; and heterogenous computers will work better together. These trends will allow more applications to work on the HP3000 allowing better integration of services from a variety of vendors over different hardware platforms.

Key Buzzwords of the 1990's: Open, Interoperability, Posix and DCE

This paper will focus on the topic of OPEN and how it relates to the HP3000. It will discuss what customers want when they describe an OPEN system and will discuss two technologies:

- o POSIX, showing the HP3000's current strength in the area of source code portability and
- o DCE, showing a key enabling technology planned in the area of interoperability.

These technologies were demonstrated at UNIX conferences held throughout the year.

What is OPEN?

Before I describe POSIX and DCE, I would first like to look at the claim of OPEN. Many computer vendors claim that they are OPEN. What does this mean precisely?

UNIX is widely regarded as being strong in the quality of OPEN. However, the industry is becoming increasingly aware that other operating systems besides UNIX can be strong in the qualities of OPEN by supporting key features.

The best way to think about OPEN is to review the fundamental requirements customers want when they think of this quality. HP surveyed customers and found that for a customer to be happy with a vendor's OPEN systems offerings it must be strong in a variety of areas. Here are the top three:

- o Application portability,
- o Multi-vendor interoperability,
- o Conformance to industry standards.

I want to discuss two aspects of OPEN that were shown at UniForum, The International Conference of Unix and Open Systems Professionals, held in January of 1992 in San Francisco:

- o Application portability as demonstrated by the POSIX demo and
- o Interoperability as demonstrated by the DCE Market Minder demo.

Both technologies are based on standards. Before discussing the demonstrations, I would like to begin with some definitions.

What is POSIX?

POSIX is an IEEE standard to support source-level application portability.

Operating systems provide services to applications running on them. POSIX.1 is the specification of an interface of how to reach these services.

Source code portability helps leverage development costs of an application since code using standard interfaces can be easily moved to other platforms that support the same interfaces.

The POSIX standard is really a set of standards in various stages of acceptance through IEEE. Here are the two standards important to this discussion:

POSIX.1	System interface standard -- defines a process and signal model, a file system, and other central services.
POSIX.2	Shell, command and utilities standard -- defines a Unix-like shell and other user utilities.

What is DCE?

DCE is a technology, planned for the future, developed by OSF that supports heterogeneous computers working together. These services are provided to make it easier to build distributed applications.

DCE makes provides a standard and consistent interface to split your application over a large, heterogeneous network.

Let's now discuss the demonstrations and technologies in more detail:

POSIX Demo

POSIX.1 and POSIX.2 were being demonstrated at the MKS booth. MKS licenses POSIX.2 technology to DEC, HP and Unisys for use on their VMS, MPE/iX and CTOS machines.

The POSIX demo had four computers side by side: DEC running VMS, Unisys running CTOS, SUN running SunOS, and HP running MPE/iX. The HP3000 and the other vendors were running a script that ran MAKE and the "C" compiler on source code for the game of Tetris. Then each computer then executed a shell script using the POSIX.2 shell. The game was then executed. A script was run creating files in a hierarchical directory. Then common UNIX shell commands were run under MPE/iX.

The same "C" source code and shell script were running on each of the four vendors' "proprietary" platforms.

The main point of the HP's POSIX demo was that by each vendor standardizing to the POSIX interface, they were able to use identical source code and achieve a standard UNIX "look and feel".

How does POSIX support Portability on MPE/iX?

Ported MPE/iX POSIX applications will perform well since POSIX on MPE/iX is an integrated solution. POSIX resides at the heart of the operating system. POSIX functionality is implemented at the same level as MPE intrinsics.

POSIX supports hierarchical directories and with long names and case sensitivity. This is useful since many operating systems use this structure, and this structure provides more flexibility than the more restrictive file/group/account organization. It is an easy matter to move files between MPE/iX and other operating systems now that MPE/iX supports hierarchical directories and long names.

Consider the following structure as an example:

[Figure 1]

Then you could enter the following command to get directory information about the above files:

```
:LISTFILE /states/WI/rivers/St_Croix
```

An example of addressing a filename by a relative path name is:

```
:LISTFILE ./rivers/St_Croix
```

assuming your current working was WI.

And:

```
:LISTFILE ./rivers/Mississippi
```

assuming MO was your current working directory.

Process management concepts with POSIX on MPE/iX have been extended to include the UNIX-style of process creation using FORK and EXEC. The FORK function creates a new process (child process) that is an exact copy of the calling process (parent process) except it has its own unique process id. The EXEC function, on the other hand, replaces the current process with a new executable.

Security has been extended to include the UNIX-like concept of a User ID and a Group ID. Each user on the system is identified by a non-negative integer known as a User ID. Each user is a member of at least one group. A group is identified by a Group ID. With these concepts filesystem access is controlled.

Posix and MPE/iX have many other important features that make application porting easier. It has a well-defined set of procedure calls to make use of operating system functions (via the POSIX.1 definition). A UNIX-like shell is available supporting the most widely used UNIX commands (via the POSIX.2 shell).

POSIX provides source code portability by providing a standard OS interface. Since many vendors provide POSIX interface support, it is an easy matter to switch among them. This will result in a wealth of new software solutions becoming available on the HP300.
POSIX provides source code portability by providing a standard OS interface. Since many vendors provide POSIX interface support, it is an easy matter to switch among them.

"If we compare a UNIX to UNIX port and a UNIX to MPE[iX] port...99.5% of the code remains unchanged."

Dr. h. c. Hasso Plattner, Vice Chairman
SAP AG

See Appendix "A" for the source of the POSIX.1 and the POSIX.2 demonstration.

Interoperability and the Market-Minder Demo

The Market-Minder Demo, also called the OSF DCE demonstration, featured a network of systems with dissimilar architectures and operating systems from various vendors, including Hewlett-Packard. Each system used DCE to run different portions of a distributed application over a network. To the user, those different platforms appeared as a single integrated system.

The demo helped users obtain timely stock market data and manipulate stock portfolios. From any system in the demo network, a user could query securities information from the Dow Jones service pertaining to a stock, and query a customer database for individual portfolio information. A heuristic simulation suggested whether it was the right time for stocks to be bought or sold. The program displayed the data with the OSF/Motif graphical user interface.

The demo went beyond connectivity, which merely provides access to data on a network; it demonstrated interoperability -- the sharing of data and compute resources among diverse systems. Different workstations in the network were designated as servers to provide the services they best perform. One system provided the customer information database, another provided the buy/sell information, and another supplied the Dow-Jones data. A user could sit at any workstation in the demo network and access the resources of all the machines. To the user, the application appeared to run as a single workstation. The demo used DCE to hide the physical complexity of the network while providing access to its resources.

The main points that the demo was that with OSF DCE:

o Users had transparent access and full use of resources anywhere in the network.

o The complexity of the network was hidden from the end users, system administrators, and application developers.

o A consistent environment was provided across different hardware platforms.

o Additional resources could be added to the network without disrupting the operation of the network.

o Tasks were distributed to the machine best suited to the task.

What is the problem DCE addresses?

DCE provides standard interfaces and services to support computers working together.

The benefits of supporting DCE are:

- More applications will become available that use client/server and other network-based models since implementing them will be consistent across many platforms.

- Data will be more easily shared and different computers will work together with each machine specializing in the task it is best suited.

- Hardware resources are used more efficiently since resources from any machine on the network can potentially be used by any user.

DCE Components

OSF's Distributed Computing Environment consists of several components which are operating system and network independent. These components are integrated to form a comprehensive set of services and tools to build distributed applications.

Fundamental Distributed Services provide tools for software developers to create the end user services needed for distributed computing. They include:

- Remote Procedure Call

- Directory Service

- Time Service

- Security Services

- Threads Service

Let's describe these services in greater detail:

Remote Procedure Call

Remote Procedure Call or RPC enables applications to call individual procedures which run on a computer elsewhere in the network. DCE RPC not only supports direct calls to procedures on remote systems, but also masks the differences between data representation on different machines, allowing programs to work across heterogeneous systems.

Distributed Directory Service

DCE's Distributed Directory Service provides a single naming model throughout the distributed environment. This model allows the users to identify by name resources such as servers, files, disks, or print queues, and gain access to them without needing to know where they are located in the network. Additionally, users can continue referring to a resource by the same name even when a characteristic of the resource, such as its network address, changes.

Distributed Time Service

Many applications need a single time reference to schedule activity and determine event sequencing and duration. Different components of a distributed application may obtain time from clocks on different computers. A distributed time service regulates the system clocks in a computer network so that they closely match each other, providing accurate time for distributed applications. The DCE Distributed Time Service is a software-based service that synchronizes each computer to a widely-recognized time standard. This provides precise, fault-tolerant clock synchronization for systems in both local area networks and wide area networks.

Security Services

Security consists of authentication for using resources, and guarantees integrity and privacy of messages sent over the network. Authentication in an open network is the verification of identity. In the network system, authentication is necessary because the messages are subject to forgery and cannot necessarily be trusted.

Authorization is concerned with granting privileges with respect to resources, such as access to files. For example, a file server must be able to determine whether a user is authorized to read a given file. One method of determining authorization is through the use of the Access Control List (ACL), a list associated with an object that describes who has what access to that object.

The DCE Security Service component provides the network with three conventional security services: authentication, authorization, and user account management. These facilities are made available through a secure means of communication that ensures both integrity and privacy.

Authentication Service

DCE's distributed Security Service incorporates an authentication service based on the Kerberos system from MIT's Project Athena. Kerberos is a trusted service that validates the identity of a user or service, preventing fraudulent requests.

Kerberos keeps a database of information on users and servers, using time-synchronized authentication tickets and private-key encryption/decryption as the trusted third-party authentication server.

Authorization Tools

After users are authenticated, they must receive authorization to use resources, such as files. Authentication is of little practical use without authorization. The Authorization facility gives applications the tools they need to determine whether a user should have access to resources.

DCE Security Services include authorization checks based in POSIX-conformant Access Control Lists (ACLs). Security services also provide a Privilege Server which forwards in a secure way the user's privileges information to the applications. This information helps the applications determine what permission it should grant to the user.

User Registry

Every computer system requires a mechanism for managing user account information. DCE Security solves the traditional problems of user account control in distributed, multi-platform networks by providing a single, scalable system for consolidating and managing user information.

The User Registry ensures the use of unique user names and passwords across the distributed network of systems and services. It maintains a single, logical database (with local replicas) of user account information including user and group naming information, login account information, and general system properties. It is integrated with the authentication service to provide a reliable and consistent user account management system.

Secure RPC

In order to use authentication and authorization services effectively, users must be able to protect the integrity and confidentiality of communications. Because messages on an open network can be read and forged, a way of determining whether a message from a given principal arrived intact and unaltered is needed. This assurance provides **message integrity**. Additionally, hiding the contents of a message from potential eavesdroppers on a network is quite often necessary in many situations. This is **message privacy**. The authenticated RPC facility within OSF DCE provides features to address both of these communication requirements.

Threads Service

Programmers want to exploit the computing power and inherent parallelism available throughout the distributed environment. The DCE Threads Service provides portable facilities that support concurrent programming, allowing an application to perform many actions simultaneously. While one thread executes a remote procedure call, another thread can process user input.

The Thread Service includes operations to create and control multiple threads of execution in a single process, and to synchronize access to global data within an application. Because a server process using threads can handle many clients at the same time, the Threads Service is ideally suited to dealing with multiple clients in client/server-based applications.

The Threads Service is used by a number of DCE components, including Remote Procedure Call, Security, Directory, and Time services.

DCE: An Integrated Environment

As we have seen from the above descriptions, DCE provides a comprehensive set of high-level, integrated services and tools for developing and running distributed client/server applications. DCE hides the complexity of the network and makes it behave as a single integrated system. The DCE components do much of the work automatically for the application developers, such as remote procedure calls and threads parallel execution. DCE also addresses problems inherent in the distributed system model, such as data consistency, privacy and time synchronization. And perhaps most important of all, DCE provides interoperability and portability of applications across heterogeneous platforms. Applications that are written on DCE can run on cooperating hosts of different architectures and operating systems, and can be readily ported to other DCE-supported hardware and software platforms.

Conclusion:

> "The message is clear: OPEN systems is not just UNIX. The HP3000 with MPE offers open systems functionality competitive with the best of systems from other vendors."
>
> Peter S. Schay, Vice President
> The Gartner Group

This article considered two key technologies that support the claim of an OPEN system demonstrated at UNIX conferences during the year: POSIX and DCE. POSIX provides important services that make porting applications easier. DCE, a technology planned for the future, gives the HP3000 important new services in the area of interoperabiltity. As the goals of OPEN evolve in the marketplace, new technologies will continue to be implemented so that the HP3000 will have the technologies customers expect in an OPEN system.

Acknowledgements

The author wishes to acknowledge Tho Le for his contribution to the DCE portion of this paper. Many of the ideas expressed in the DCE portion of this paper are considered in an "Introduction to OSF DCE", December 31, 1991 from the Open Software Foundation, 11 Cambridge Center, Cambridge, MA. A special thanks to Rick Bartlett and Dennis Lee who reviewed early drafts of this paper and provided much helpful criticism.

Appendix A

```
function dem {
banner <<\eof


Posix.1
Demo
eof
sleep 3

echo 'Posix.1 (IEEE Std 1003.1 or ISO/IEC 9945-1) defines a
programmatic echo 'interface to the operating system. In this
demonstration, we build' echo 'a program to play the game Tetris
which uses these interfaces.\n' echo '(As an aside - tetris uses
the curses package which is standardized' echo 'by X/Open.)'
sleep 10

echo 'Here is a makefile for the tetris program:\n'
echo '$ cat tetris.mk'
cat tetris.mk
sleep 6

echo '\nNow make the program:\n'
echo '$ make -fu tetris.mk'
make -f tetris.mk
sleep 6

echo '\nand finally run the resulting program:\n'
echo '$ tetris'
sleep 3
tetris

sleep 3

banner <<\eof

Posix.2
Demo
eof
sleep 3

echo 'Posix.2 (IEEE 1003.2) defines a set of utilities and a
command language' echo 'that allows the user to manipulate the
computer to accomplish the' echo 'desired task. The following
demonstrations illustrates the use of' echo 'these utilities to
accomplish a variety of everyday tasks.\n' sleep 12

echo 'Let'\''s create a subdirectory to contain our work:\n'
echo '$ mkdir work'
mkdir work
```

```
sleep 4
echo '\nnow move into that directory:\n'
echo '$ cd work'
cd work
sleep 4

echo '\nlet'\''s see where we are:\n'
echo '$ echo $PWD'
sleep 4
echo $PWD

sleep 4
echo '\nLet'\''s build some files in this directory using the awk
programming' echo 'language. We will build three files, each with
one line of text. Here is' echo 'the program:\n'
cat <<\eof
$ awk 'BEGIN {
     for (i=1; i<4; i++)
          print "This is testfile " i > ("test" i)
}'
eof
sleep 4
awk 'BEGIN {
     for (i=1; i<4; i++)
          print "This is testfile " i > ("test" i)
}'

sleep 4
echo '\nLet'\''s see what files we'\''ve created, using the "ls"
command:\n' echo '$ ls'
sleep 4
ls

echo '\nWe can also use the echo command with wild card expansion
to do the' echo 'same thing:\n'
echo '$ echo *'
sleep 4
echo *

sleep 4
echo '\nOf course the ls command can give us a lot more
information:\n' echo '$ ls -l'
sleep 4
ls -l

sleep 4
echo '\nUse the cat command to look at the contents of these
files\n' echo '$ cat *'
sleep 4
cat *
```

```
sleep 4
echo '\nThe diff utility is used to compare 2 files:\n'
echo '$ diff test1 test2'
sleep 4
diff test1 test2

echo '\nAnd the grep utility can be used to search files for a
pattern:\n' echo '$ grep 2 *'
sleep 4
grep 2 *
sleep 4

echo '\nHere is a more complicated example using pipes. The output
of the' echo 'ls command is fed into an awk program which converts
all of the file' echo 'names into upper case. The output from awk
is then fed into banner' echo 'which prints them using large
characters. (The banner utility is\n' echo 'not part of Posix - it
is defined by X/Open.)'

cat <<\eof
$ ls -l ¦ awk '{print toupper($NF)}' ¦ banner
eof
sleep 10
ls -l ¦ awk '{print toupper($NF)}' ¦ banner
sleep 4

echo 'We'\''re done our work, so let'\''s clean up. First remove
all of the' echo 'files in this subdirectory:\n'
echo '$ rm *'
rm *
sleep 4

echo '\nthen move up to the parent directory:\n'
echo '$ cd ..'
cd ..
sleep 4

echo '\nand remove the '\''work'\'' subdirectory:\n'
echo '$ rmdir work'
sleep 4
rmdir work

echo '\nThat'\''s all for now!'
}

while :
do dem
done
```

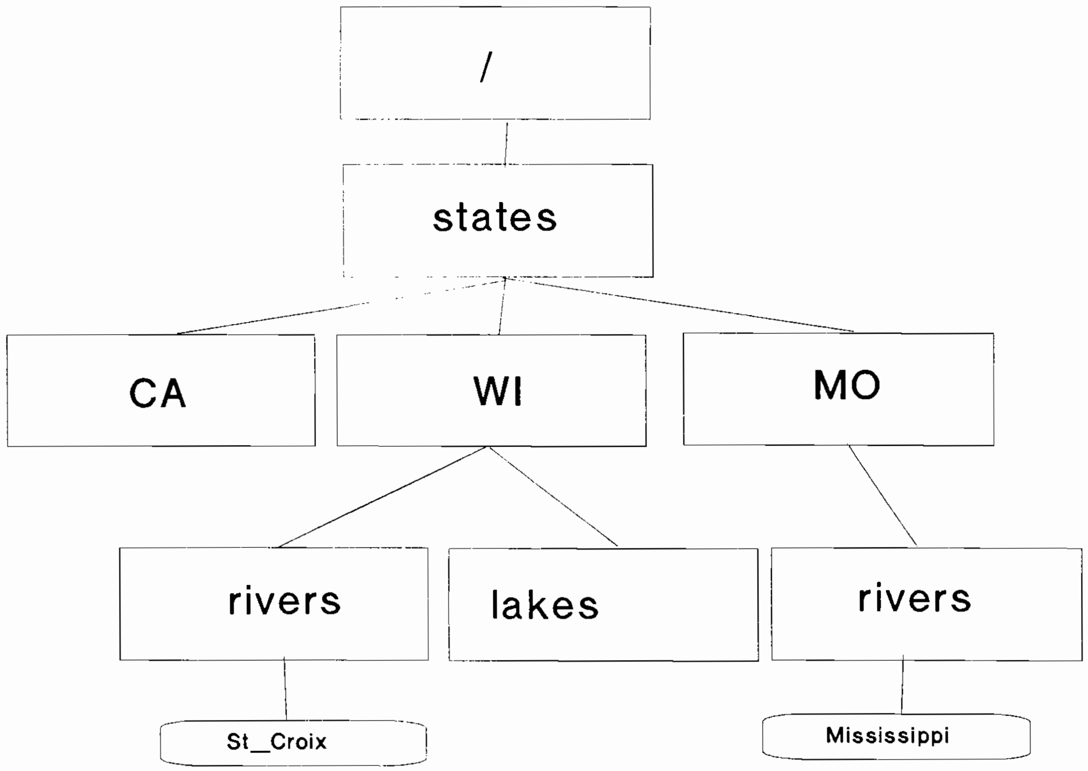


Figure 1

WHAT DOES MIGRATING TO ALLBASE/SQL MEAN FOR YOU?

PAPER #3807

Janice Hodor
Hewlett-Packard Company
2205 East Empire Street
Bloomington, IL 61704
(309) 662-9411

More businesses are beginning to implement relational database applications in their environments. Your success at migrating to ALLBASE/SQL can be greatly enhanced by understanding the key ALLBASE/SQL concepts. If you are migrating from TurboIMAGE to ALLBASE/SQL, you'll gain additional benefit from being aware of some of the specific differences between the two databases. This paper will provide the key ALLBASE/SQL concepts as well as define some of the differences between ALLBASE/SQL and TurboIMAGE. This information is intended to be used by your organization's development and support staffs to simplify the migration to ALLBASE/SQL.

One of the first differences you'll encounter when migrating from a TurboIMAGE environment to an ALLBASE/SQL environment is the database terminology. Some of the other differences include ALLBASE/SQL's flexible database naming convention and the files generated at creation time for an ALLBASE/SQL DBEnvironment.

ALLBASE/SQL Terminology

In ALLBASE/SQL, databases are located inside a structure known as a DBEnvironment (database environment or DBE). A DBEnvironment can actually contain one or more databases. The number of databases in a DBEnvironment is dependent upon the amount of available disc space. A DBEnvironment is similar to TurboIMAGE's "root file" concept with the exception of having the capability for multiple databases to share the same root file.

The data in a relational database is organized in tables. A table is a two-dimensional structure of columns and rows. A row is similar to an entry in a

TurboIMAGE dataset and a column is similar to a data item or field in a TurboIMAGE dataset. ALLBASE/SQL defines a database as a group of tables having the same owner. The owner has special capabilities such as the ability to delete a table and the ability to transfer ownership to someone else.

A "view" is a logical table derived by placing a "window" over one or more tables to let users or programs view only certain data. Views are used to improve security by allowing users to access only that data for which they have a need and to store complex frequently used SELECT statements. A view is a SELECT statement stored in the system catalog, not a physical copy of the data; therefore, views do not result in redundant data. ALLBASE/SQL extracts data from physical tables at the time you use the view.

Figure 1 illustrates the concept of a table, row, column and view.

TABLE1

NAME	DEPT	SALARY	BIRTHDAY
Ann	MIS	22,000	10/04/62
Bill	ACCT	80,000	07/16/47
Bob	MKTG	65,000	03/10/54
Mary	MIS	35,000	02/08/59
Sue	MKTG	20,000	07/04/56

Row

Column

VIEW1

NAME	BIRTHDAY
Ann	10/04/62
Mary	2/08/59

```
CREATE VIEW VIEW1
AS SELECT NAME, BIRTH
FROM TABLE1
WHERE DEPT = 'MIS';
```

Figure 1

Before you can create tables and load data into a database, you need to allocate physical storage space by creating DBEFiles. DBEFiles are nothing other than privileged MPE files that reserve disc space for your data and indexes. Data in logical databases is stored in one or more DBEFiles. Indexes for a table are also stored in DBEFiles. Most of the files associated with a DBEnvironment are DBEFiles. There are three types of DBEFiles.

- o TABLE - contains only table data
- o INDEX - contains only index data
- o MIXED - can contain both table and index data

In addition to specifying the DBEFile's type, you also need to specify the size of the DBEFile (in number of pages; a page is defined as 4096 bytes).

These DBEFiles are then logically grouped together to create a DBEFileSet. Once the DBEFileSets have been created, you then associate the DBEnvironment's tables with the DBEFileSets. For ease of maintenance and performance tuning reasons, you can have a variable number of DBEFileSets associated with a DBEnvironment.

Since a DBEFileSet is a logical grouping, it does not have a specific capacity. To increase a DBEFileSet's capacity, you simply add DBEFiles to a specific DBEFileSet.

The tables and indexes associated with a DBEFileSet do not have to be for the same logical database. Figure 2 illustrates that while a DBEFileSet can contain data for all the tables in a database, a DBEFileSet can also contain data for some of the tables in a database, or for tables in more than one database. Thus DBEFileSets offer a way to allocate data storage independently of how users think about the data.

Since the DBEnvironment consists of several MPE files, the person creating the DBEnvironment has control over the naming convention. This differs greatly from TurboIMAGE in that the DBEFiles can be named any related or non-related name.

For example, when you created a database in TurboIMAGE, you specified the root file name in the database's schema and TurboIMAGE automatically created the dataset names by appending two alphanumerics to the root file. This provided an intangible benefit when it came to trying to see all the files associated with a given database using the LISTF (list file) command.

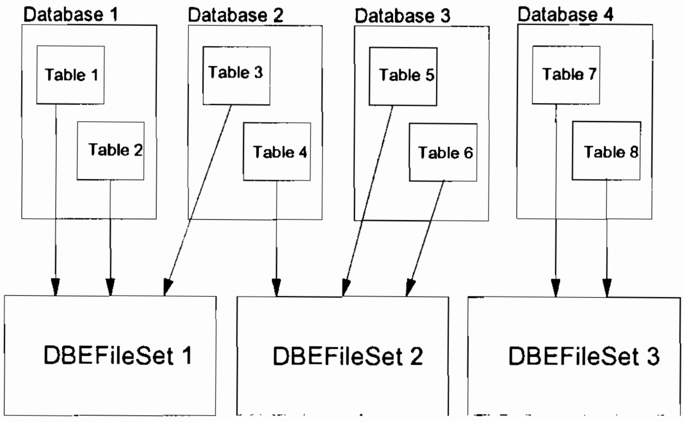


Figure 2

When you create an ALLBASE/SQL DBEnvironment, three physical files are generated - DBECon, the DBEnvironment's log file, and DBEFile0. The DBEnvironment Configuration file, or DBECon file, contains basic information that is used every time the DBEnvironment is opened, e.g. log mode, log file name, etc. The DBEnvironment's log file is used for archive or non-archive logging. DBEFile0 allocates space for the system catalog. The system catalog is reserved for information about the DBEnvironment (tables, indexes, etc.).

Unlike TurboIMAGE, ALLBASE/SQL is very flexible in how these physical files are named. You have the ability to implement your own 8-character naming convention for the DBECon file, DBEnvironment log file and all DBEFiles. These are the files you will see when you perform a LISTF (list file) command.

If you use third party backup solutions, you may find it helpful to implement a naming convention similar to TurboIMAGE's naming convention. If PARTSDBE was the name of your DBEnvironment, you could name your files as follows.

PARTT000	Table data - "T"
PARTI001	Index data - "I"
PARTM002	Mixed data - "M"
PARTLG01	Archive log file 1
PARTLG02	Archive log file 2

Why customers are considering HP ALLBASE/SQL

ALLBASE/SQL is a strategic database for Hewlett-Packard platforms. Hewlett-Packard customers are considering HP ALLBASE/SQL for the following reasons.

1. Easy to use/learn/design/maintain
   - Results in lower training and maintenance costs

2. Flexible data structures
   - Dynamic database restructuring
   - Supports large data structures for storing graphical data and images

3. Program and data independence
   - You can make changes to the DBEnvironment without having to recompile your programs

4. Standard
   - HP's support for leading industry standards

5. Open
   - A choice of leading multi-vendor application development tools
   - Allows users to mix and match heterogeneous SQL products in a multi-vendor environment

6. Tight integration with the operating system
   - Allows maximum use of available CPU power and operating system features
   - Results in leading on-line performance on HP platforms

7. Tools
   - Complete set of HP and multi-vendor tools for application development and connectivity

8. High availability
   - Supports on-line backup, automatic log switching, dual logging, and dynamic space expansion

9. Leading OLTP performance on HP
   - Fastest RDBMS on HP platforms (based on audited and fully disclosed TPC-A and TPC-B benchmark data)

10. Leading price/performance

ALLBASE/SQL Utilities

In any database environment, it is helpful to have a set of tools that assist in creating and maintaining the database. These tools or utility programs are used primarily by the database administrator (DBA) but also benefit the development and support personnel.

There are several DBA utilities that are available through either third party vendors or Hewlett-Packard. When you purchase ALLBASE/SQL you automatically receive four DBA utility programs:

- o ISQL
- o SQLGEN
- o SQLUtil
- o SQLMigrate

ISQL (Interactive Structured Query Language) functions as an interactive command processor that lets users enter SQL commands at the keyboard and observe query results and other information. It is primarily used by the DBA to create and modify the schema of an ALLBASE/SQL DBEnvironment. ISQL offers a mixture of the capabilities provided in TurboIMAGE's QUERY, DBSCHEMA, DBLOAD, DBUNLOAD, and DBUTIL utilities.

Application developers can use ISQL as a query utility, e.g. give me all the part numbers that meet some specified criteria (similar to TurboIMAGE's QUERY product). ISQL can also assist application developers in the testing of complex queries (inserts, updates and deletions can also be performed within ISQL) before actually coding them into the application. This saves them the overhead and time involved in recompiling the program before verifying the query's actual results.

ISQL is also used for LOADing and UNLOADing table data. Some of the uses of the LOAD/UNLOAD capability are adding and/or deleting column(s), merging two tables together or copying your DBEnvironment to another system (e.g. development vs. production). ISQL's LOAD/UNLOAD feature is similar to the extract/import capability provided by Robelle's "SUPRTOOL" product for TurboIMAGE databases.

ISQL command files offer a shortcut to creating databases. Command files allow you to store a series of ISQL and SQL commands and then, with a single START command, execute all the commands in that file. This is very useful for groups of commands you execute together frequently. In addition, if an entire DBEnvironment is created by using command files, it is easy to recreate the DBEnvironment, as well as examine or modify any part of its definition. The SQLGEN utility can be used to create the command files that are used by ISQL.

SQLGEN is a utility that generates the commands used to recreate all or part of an existing ALLBASE/SQL DBEnvironment. These commands are placed in one or more data files called schema files which can then be used as ISQL command files to recreate the DBEnvironment.

SQLGEN has several uses. DBEnvironments can be designed and tuned on a development system and then the entire schema can be transferred to a production system. SQLGEN can also help move part of a DBEnvironment. For example, if a department transfers from one site to another, that department's DBEnvironment structure can be easily moved to a new system.

SQLUtil is a utility program for database administrators that assists in displaying and setting the basic parameters of a DBEnvironment, storing and restoring DBEnvironments, setting the size of system buffers, and purging DBEnvironments. SQLUtil is primarily used by the DBA. SQLUTIL offers a mixture of the capabilities provided in TurboIMAGE's DBUTIL, DBRECOV, DBSTORE and DBRESTOR utilities.

Support personnel may need to use SQLUtil to move DBEFiles or log files from one location to another. As of ALLBASE version F.0, you can move the DBEnvironment's log file to a volumeset other than the one where the DBEnvironment resides. SQLUtil would also be used to perform rollforward recovery.

SQLMigrate is a utility program for database administrators that assists in migrating a DBEnvironment from one version of ALLBASE/SQL to another without unloading and reloading data. This utility is both forward and backward compatible. SQLMigrate is equivalent to TurboIMAGE's DBCONV utility.

The concept of SQLMigrate is similar to that of migrating from IMAGE to TurboIMAGE. In order to update the database's internal data structures, you were required to run a migration utility.

ALLBASE version E.0 currently runs on MPE/iX release 2.2. If you upgrade your operating system to MPE/iX release 4.0, you would need to run SQLMigrate with the "forward" option against each DBEnvironment. This will allow the DBEnvironment's internal structures to be compatible with the new ALLBASE/SQL release. Similarly, if you installed a bad patch or wanted to revert to a previous release of the operating system, you would need to run SQLMigrate with the "backward" option against each DBEnvironment.

Capacity management includes making sure that there is enough room to hold the DBEnvironment's data and if there is too much room, to free up the extra disc space. Capacity management is primarily performed by the database administrator.

In the TurboIMAGE world, most HP customers use a utility such as Adagers's "Adager" product to expand and contract a data set's capacity. With ALLBASE/SQL, you are given the ability to expand a DBEFileSet's capacity.

A DBEnvironment's capacity is increased by using one of two options: 1) manually creating another DBEFile and adding it to the DBEFileSet or 2) defining a DBEFile as being expandable through the use of an ALLBASE F.0 feature called dynamic space expansion. Dynamic space expansion allows the DBEnvironment system to increase the DBEFile space when needed without any user intervention.

Query Access Plan Display provides a new command, GENPLAN, to be used via ISQL or a pre-processed application running ALLBASE version F.0. GENPLAN is given the query and returns with the access plan, which it places in the system temporary table SYSTEM.PLAN. The table disappears at the end of the transaction.

By viewing the contents of SYSTEM.PLAN, the application developer will be able to understand the steps and the operations involved in the query. That user can then formulate the query in a different manner or make other design changes in anticipation of reducing the overhead and thus increasing the performance of the query.

Programming Impact

When developing ALLBASE applications, there are a few concepts that are different than TurboIMAGE. These concepts include transaction management, locking, and the optimizer.

A transaction is one or more SQL commands that together perform an application-dependent operation on one or more databases in one DBEnvironment. Simple transactions may accomplish only one operation. Some transactions may accomplish several operations.

The application developer is responsible for defining transactions within the application. The BEGIN WORK command is used to begin a transaction. The COMMIT WORK is used to end a transaction. This is similar to TurboIMAGE'S DBXBEGIN/DBXEND concept. Transactions do not take effect until they are committed.

There are times when your application may encounter an error condition and you won't want to commit the effects of one or more commands. The ROLLBACK WORK command will roll back all changes within a transaction and end the transaction.

To roll back partial transactions, use a savepoint. A savepoint is a marker you can place between commands using the SAVEPOINT command. Later in the transaction you can roll back work to a specific savepoint by using the ROLLBACK WORK command with the TO option. After you issue the SAVEPOINT command, ALLBASE/SQL returns a savepoint number. Use this number to do a ROLLBACK WORK. Savepoints are useful for long transactions where you don't want to roll back the whole transaction and resubmit the transaction.

Locking is a technique for concurrency control through which ALLBASE/SQL restricts access to data by one individual when the data is being used by another. There are three types of locks: shared, exclusive, or shared with intent to become exclusive. Lock type is determined by the type of table being accessed and by the kind of operation the user is performing. Locks are released when a transaction ends with a COMMIT WORK command.

Unlike TurboIMAGE, ALLBASE/SQL does offer automatic deadlock detection. When a deadlock is encountered, the current transaction is automatically rolled back. The transaction must be captured and re-applied in order for it to take place.

From a coding perspective, one of the biggest differences from TurboIMAGE is that ALLBASE/SQL does not force you to code the access path to get at the data. ALLBASE/SQL provides an optimizer which takes query commands and determines the best way to access the data that's a part of that command. As an application developer, you simply tell ALLBASE what data you want and what you want to do with it (SELECT, INSERT, UPDATE or DELETE) and it takes care of the way the information is obtained for you.

Logging

At the time you create a DBEnvironment, ALLBASE/SQL sets up a log to record DBEnvironment activity. While the DBEnvironment is running, those SQL commands that modify data are tracked in log records which are eventually stored in one or more physical log files associated with the DBEnvironment. Unlike TurboIMAGE, logging is required with ALLBASE/SQL.

ALLBASE/SQL log records are used primarily to maintain database integrity. Some of the specific uses include:

- o Recovering a DBEnvironment to a consistent state after a system crash. Updates of all committed transactions are guaranteed to be recovered. Incomplete transactions are rolled back (undone), hence the name rollback (or soft crash) recovery.

- o Recreating the DBEnvironment from the previous backup copy up to the current time in the case of media failure on the DBEFile(s). This is a rollforward recovery.

- o Rolling back user transactions which cannot be successfully completed.

Since logging is provided at the DBEnvironment level, this implies that recovery is performed at the DBEnvironment level. Keep this fact in mind when designing your DBEnvironment.

ALLBASE/SQL offers two kinds of logging: non-archive and archive logging. Non-archive logging tracks all current DBEnvironment activity into a circular log file. Once a transaction is committed, the space that that transaction occupied can be reused by other log records. Non-archive logging is the default in ALLBASE.

Archive logging tracks all DBEnvironment activity continuously from the last DBEnvironment backup. This type of logging is used in conjunction with rollforward recovery.

ALLBASE currently supports logging to disc, not tape. When you create the DBEnvironment, you specify either SINGLE or DUAL logging. In single logging, ALLBASE/SQL maintains one set of log files in either archive or non-archive mode. For greater security, you may want to specify dual logging which duplicates a set of log files. In dual log mode, committed transactions are written to both log files at the same time. ALLBASE version F.0 provides the ability to store archive log files on a volume set other than where the DBEnvironment is located.

If you have a very large transaction that would cause the non-archive log file to overflow, the transaction will be rolled back (undone) completely and will need to be re-submitted. You will need to either create or add a new log file that is large enough to accommodate such transactions.

If you have a transaction that would cause an archive log file to overflow, you can create another log file to prevent the overflow situation. You should store the log files as they fill up. Once stored, and once all the transactions in them are complete, archive log files are marked available for reuse.

For most ALLBASE customers, non-archive logging is appropriate for the phase of database creation and initial loading of tables. Once the DBEnvironment has been loaded, you can either use ISQL's "START DBE NEWLOG" command or use SQLUtil's STOREONLINE command to create a complete backup and turn on archive logging. The STOREONLINE option is only valid if you have the TurboSTORE/iX product.

Both TurboIMAGE and ALLBASE provide archive logging. ALLBASE's archive logging is at a very physical level, e.g. one log record is written with the previous value of the data (a before-image) and a second log record is written with the new value of the data (an after-image). Before-images are used to remove the results of incomplete transactions. After-images are used to make the results of completed transactions persistent. TurboIMAGE logs one record for each modification to the database.

As a result of the physical nature of ALLBASE's archive logging implementation, the log file format can change (and does!) from one version of ALLBASE to the next. This is very different from TurboIMAGE!

Backup options

Backup and recovery are activities that let you reconstruct your DBEnvironments in the event of database corruption or damage to your system. The DBEnvironment backup will be used in conjunction with archive logging.

There are four different backup options for ALLBASE:

- o Static
- o Concurrent
- o Third Party
- o ALLBASE/Replicate

A static backup requires stopping the DBEnvironment. This means that you cannot have any users logged on to any applications which utilize the DBEnvironment. SQLUtil's STORE command is used to make a copy of the DBEnvironment, including the DBECon file. If you are using archive mode logging, SQLUtil's STORELOG command would be used to back up the logfiles to tape. Once the static backup copy is complete, the database administrator would then start the DBEnvironment for production activity.

Concurrent backups get their name from the fact that users can continue working during the backup process. SQLUtil's STOREONLINE command is used to back up the entire DBEnvironment. If you are using archive mode logging, SQLUtil's STORELOG command would be used to back up the logfiles to tape. Concurrent backups require the use HP's TurboSTORE/iX product with the on-line option.

Third party backups are essentially a "static" backup performed with third party products, e.g. Hi-Comp's "DBTune", Tymlab's "BackPack", etc. Today, the third party solutions do not support the on-line backup feature. Check with your third party vendor to see if their product will backup the entire DBEnvironment. Even if you use a third party backup solution, SQLUtil's STORELOG command is the only supported method for backing up log files.

ALLBASE/Replicate provides a set of intrinsics that are used to implement a shadowed DBEnvironment (similar to Silhouette for TurboIMAGE databases). You can choose to replicate the entire DBEnvironment or a subset of a DBEnvironment's data. ALLBASE/Replicate allows the replicated DBEnvironment to become the primary DBEnvironment if the primary DBEnvironment fails.

ALLBASE/Replicate does require extra logging to generate audit trail records which contain table name, column numbers, inserted or updated values, etc. of committed transactions. These committed transactions are then applied against the replicated DBEnvironment (or subset). The timeliness of the replicated DBEnvironment is dependent upon when the audit trail records are applied to the replicated DBEnvironment. The replicated DBEnvironment can then be used for offloading some of the processing which would normally be performed against the primary DBEnvironment, such as ad-hoc queries, report generation, etc.

Conclusion

ALLBASE/SQL continues to be a strategic database for Hewlett-Packard platforms. ALLBASE/SQL's openness, multi-vendor tools, flexible data structures, database administrator utilities and high-availability features enhance ALLBASE/SQL's performance.

This paper highlighted the key differences between ALLBASE/SQL and TurboIMAGE from an application development and support perspective. Now that you are aware of these differences, you can begin to develop a plan to address these in your environment.

The typical successful ALLBASE/SQL customer sites have attended ALLBASE/SQL customer education, invested time up-front in their database design(s), spent less time programming the application(s), and have tools available to support their database. Bottom line, they are able to develop and maintain ALLBASE/SQL applications in less time which results in lower costs.

Basic Dump Analysis For System Managers
Paper #3808

David Kohl
Hewlett Packard
2000 South Park Place
Atlanta, Georgia 30339
(404) 953-1840

The HP3000 systems are well positioned for the future in terms of software resiliency. The MPE/iX operating system is more robust, more resilient to failure, less likely to fail, and more able to correct problems that have previously resulted in failures before. In the unfortunate case that your system does fail, however, would you not like to have the knowledge and tools to gather valuable data about the failure?

Once you are familiar with the tools used to look at a memory dump, there is a lot of useful information that you can find yourself. This information can help you look for trends and common problems.

The tool used to analyze memory dumps is called DAT (*Dump Analysis Tool*). Most of the commands and features of DAT are also available in the System Debugger (*DEBUG*). Unless otherwise noted, everything in this paper is applicable to DEBUG as well as DAT.

To begin, we will look at some of the basic features and functions of both DAT and DEBUG. The intent is to familiarize ourselves with how powerful these tools are so that you can proceed on your own to find other uses for the tools than simply looking at memory dumps. We will complete our discussion with some suggested steps for analyzing system aborts and system hangs.

I. Performing a Memory Dump

Before we get into the discussion of reading a memory dump, it is important that we understand how to properly perform a dump and also how to load the dump to disk. If this is an operation that you are already familiar with you may proceed to Section II (Basic Commands).

When an MPE/iX system fails, you will see a message on the console similar to this :

```
SYSTEM ABORT 1458 FROM SUBSYSTEM 128
SECONDARY STATUS, INFO=128, SUBSYS=101

SYSTEM HALT 7, ($0FB4)
```

To perform the memory dump go into control mode on the system console by typing a CONTROL-B (hold down the Ctrl key and press B). You should get a CM> prompt. There are two commands that will restart the system at this prompt, RS (*Reset*) and TC (*Transfer Control*). The difference is that an RS will clear all memory, rendering a memory dump useless. It is important that you restart the system with a TC.

After doing a TC, boot the system from primary path, interacting with IPL, and type DUMP at the ISL> prompt. If you have autoboot enabled on your system, remember that you will need to override the autoboot process.

The complete process should look like this :

```
<CNRL-B>
CM> tc

Autoboot enabled, hit any key within 10 seconds
<press the space bar to interrupt the boot process>

Boot from primary path (Y/N)? Y

Interact with IPL (Y/N)? yes

ISL> dump
```

Be sure you have a scratch tape with a write ring mounted on your alternate boot path prior to typing "dump". After typing the dump command you will be asked to enter an 80 character dump string which is simply a free-form title for the dump. After the dump is complete, you can reboot the system with a normal START RECOVERY.

Once the system has been rebooted, you must copy the memory dump from tape to disk in order to read the dump. This is done with a utility called DAT. To load the dump, perform the following steps with the dump tape mounted on any tape drive :

```
:HELLO MGR.TELESUP,DAT
:DAT
nmdat> GETDUMP dumpfilename
```

The '*dumpfilename*' can be any five character file name. The actual dump file will have the characters MEM appended to the end.

II. Basic Commands

DAT offers a very powerful command set for displaying basic information. Some of the basic commands you should be familiar with are listed in Table-1.

DAT/DEBUG Commands

pin	select process to monitor
tr,i,d	full stack trace of process
cm & nm	switch between modes
dv	display virtual data location
w & wl	write & writeln for output

Table-1

In addition to these basic commands, DAT has a set of window commands. The windowing functionality allows you to keep a set of default information displayed on the screen with continuous updating. The basic windows for a native mode process will display all registers and a procedure window showing the current code location and instruction. Other windows can be added to track data and code locations. Familiarity with the windows facility is not necessary to obtain valuable information from a dump and will not be discussed any further in this paper. Some basic window commands are listed in Table-2 but for more detail refer to Chapter 7 of the Debug Reference Manual.

DAT/DEBUG Window Commands

won & woff	turn all windows on and off
vw	enable a virtual data window
pf & pb	scroll through program window
vr	change display mode of vw

Table-2

III. DAT/Debug Macro Facility

DAT and DEBUG both have an elaborate macro facility. Macros are a sequence of commands that can be executed by a single name. MPE/iX systems come with a standard set of macros which reside in a file called MACSTART.DAT.TELESUP. Whenever you are looking at a dump or a live system, you should start by loading these macros. Figure-1 shows the syntax for for doing this in DAT. The syntax would be similar in DEBUG.

```
:debug
 $1 ($5f) nmdebug > use macstart.dat.telesup

Welcome to the DAT Macro facility.

Please choose which macros & symbol files you wish to load. Multiple choices
may be entered all at once (default <cr> is all). Valid input examples: "1 2 3"
or "OS DB" or "1,DC" or <cr> or "ALL".

   1) OS macros & symbols
   2) Data Comm macros & symbols
   3) Data Base Core & Turbo Image macros & symbols
   8) User macros & symbols
   9) Do not load any macros

Enter your selection now: 1

$2 ($5f) nmdebug >
```

Figure-1

In addition to the system macros, customized macros can be defined using the macro command. This can be done interactively or by creating a macro file. When defining a macro, you must supply the name and the body of commands. Parameters can optionally be specified. Following is an example of a one command macro that would be used to display the current time :

```
$nmdat> macro showtime {wl 'Current Time :  ' time}
$nmdat> showtime
Current Time :  9:43 AM
```

More complicated macros that execute several commands are best defined in a separate text file. The macros can then be loaded using the USE command, specifying the macro file name. Figure-2 is an example of a macro that accepts a pin number and dumps some basic information for the process. This macro could be created using any text editor.

```
mac pin_info (this_pin_:s32)
{
/*  Macro Name : pin_info
 /*  Author     : Joe Bedutz
 /*  Date       : 01/23/92
 /*
 /*  Parameters : this_pin : S32 -
 /*
    wl;
    wl  'Information for pin : ',this_pin;
    wl;
    pin !this_pin;
    pm_family(!this_pin);
    tr,i,d;
    wl;
 }
```

Figure-2

To load and execute the macro in Figure-2 you would perform the following commands :

```
$nmdebug> use macfile
$nmdebug> pin_info
```

Macros that you define can execute any DAT/DEBUG command as well as any other macro, providing the other macro has already been loaded.

IV. Formatting Data Structures

Some of the most useful information in a memory dump comes from the interpretation of data structures. DAT/DEBUG offers a symbolic formatter which is a powerful way of displaying data with a defined type. Though you may create your own data structure definitions with any language, all data interpretation is performed as if it were a Pascal data type. Therefore, Pascal will be used in all examples in this paper.

Consider an example of running Debug against the program whose compiled listing is shown in Listing-1 (see page 7). When you run this program with debug and break at an address after the initialization of student_record, you could display the raw data in the student_record as follows :

```
($8b) nmdebug > dv dp+10,d
VIRT $8a0.40331018 $ 42656475 747a2020 20202020 20202020
VIRT $8a0.40331028 $ 20202020 4a6f6520 20202020 20200013
VIRT $8a0.40331038 $ 0000005d 00000000 00000057 00000000
VIRT $8a0.40331048 $ 00000000
```

The data will not have much meaning without first going through the manual conversion of each data location to its proper data type. Using the symbolic formatting facility will allow us to display the data in a more natural way.

To generate your own symbolic data types for use in DAT/DEBUG, you will need to include the $SYMDEBUG 'xdb'$ option at compile time. If you are not using Pascal refer to the appropriate language reference manual for the option needed. This option causes symbolic debug records for code and data to be written to a relocatable library. DAT/DEBUG only uses the data type definitions but the body of the source being compiled still must include at least one simple statement. Listing-2 (see page 8) shows the source for a file that will be compiled to generate a symbolic data type for the student_record in the prior example.

```
0    1.000  0  $CODE_OFFSETS ON$
0    2.000  0  $TABLES ON$

0    3.000  0  program student (input, output);
0    4.000  0
0    5.000  0  type
0    6.000  0    student_type = PACKED RECORD
1    7.000  0                 last   : packed array [1..20] of char;
2    8.000  0                 first  : packed array [1..10] of char;
3    9.000  0                 age    : shortint;
4   10.000  0                 grades : array [1..5] of integer;
5   11.000  0                END;
5   12.000  0
5   13.000  0  var
5   14.000  0    i : integer;
6   15.000  0    student_record : student_type;
7   16.000  0
7   17.000  1  begin
7   18.000  1
7   19.000  1    student_record.last:='Bedutz';
8   20.000  1    student_record.first:='Joe';
9   21.000  1    student_record.age:=19;
10  22.000  1
10  23.000  1    for i:=1 to 5 do student_record.grades[i]:=0;
12  24.000  1    student_record.grades[1]:=93;
13  25.000  1    student_record.grades[3]:=87;
14  26.000  1
14  27.000  1    writeln('Student Record Initialized');
15  28.000  1
15  29.000  1  end.
```

Listing-1

To get the xdb source file into symbolic format, you will need to compile and link the program as you normally would and then run the program file through a preprocessor and preparation stage. The following steps create the symbolic definitions for the student_record :

```
:PASXL SOURCE,OBJECT,$NULL
:LINK FROM=OBJECT;TO=STUDENTP
:PXDB.PUB.SYS STUDENTP
:DAT
$nmdebug>  SYMPREP STUDENTP
```

```
$SYMDEBUG 'xdb'$
 program studtyp (input, output);

type   student_type  = PACKED RECORD
                last   : packed array [1..20] of char;
                    first  : packed array [1..10] of char;
                age     : shortint;
                grades : array [1..5] of integer;
                END;

var   x : integer;

begin
  x:=1;
end.
```

Listing-2

Now when you run the program and break the process at the same address, you can format the data in a more readable format. This is done by opening your new symbol file and using the fv (Format Virtual) command.

```
($5f) nmdebug  >  symopen symstud
$7 ($5f) nmdebug  >  fv dp+10 'student_type'

PACKED RECORD
  LAST   : 'Bedutz
  FIRST  : 'Joe
  AGE    : 13
  GRADES :
    [ 1 ]: 5d
    [ 2 ]: 0
    [ 3 ]: 57
    [ 4 ]: 0
    [ 5 ]: 0
 END

($5f) nmdebug>
```

The prior information explains how you can use symbol files to assist in debugging your own programs. The same functionality is provided by DAT for use in dump analysis. The MACSTART script opens a symbol file called SYMOS that contains data definitions for the operating system. Once you have opened a symbol file (such as SYMOS) you can see the data definitions available to you using the syml command. For example, if you want to see all the data types that contain the string "LABEL" you can do this as follows :

```
        $nmdebug> symopen symos.osa51.telesup
        $nmdebug> syml @LABEL@
```

You can then proceed to format the desired data address with one of the data types found.

V. Basic Dump Analysis

Now that we have become familiar with the DAT/DEBUG facilities, we are ready to move into the details of dump analysis. We will do this in three parts, beginning with a discussion of some general information commands and macros to be used with all types of dumps. We will then discuss some specifics for system aborts and system hangs.

General Information

There are several pieces of general information that you will want to process in all types of dumps. Most of this information comes from macros defined in MACSTART.

The machine_state macro gives a general overview of the system state at the time the dump was taken. The output from this macro is shown in Figure-3.

Machine_state tells you whether system_abort was called, the pin number of the current process(likely the one that called system_abort), and dumps all registers. The individual pieces of information could be retrieved from the macros process_current (returns the pin number of the current process) and dr (display registers).

Once you have a pin number for the current process (process running at the time of the dump) you will want to get the actual name of the process. This can be done with the command dptree (pm_ptree in DEBUG) or pm_family.

The benefit of pm_family is that it will display information about the parent process if a process is child.

```
$140 ($0) nmdat > machine_state

SYSTEM ABORT NOT CALLED.
MPE/XL VERSION: B.08.14          Current CPU Last Pin : $7ffd

SYSTEM CONSOLE AT LDEV #20

Processor:  00  HPA: fff80000  IVA: 00145000  Config: TRUE
Current CPU: 0  Original CPU: 0  Monarch CPU: 0  MP array at: 6f3000


        CPU($0):     DISPATCHER_PAUSED    Last Pin : $7ffd

HP3000 SERIES 949 With Processor Revision 0.

CURRENT REGISTERS:

                        < General Register Listing >

                        < Special Register Listing  >
```

Figure-3

Once you know the pin number for the current process, one of the most important pieces of information that you can display for any type of dump is a process trace. A process trace gives a good picture, through procedure names, of what a process was doing at the time of the dump. Following is an example of a partial trace for a process performing an FOPEN of a KSAM file.

```
   CM  * 0) SYS  % 131.2225   FSETMODE+%75
   CM    1) SYS  % 156.2771   KOPEN+%1533
   CM    2) switch marker
 NM  c) SP=4035d5e8 RP=a.004916e4  switch_to_cm+$86c
 NM  d) SP=4035d3f8 RP=a.004c0618  kopen_+$1c4
 NM  e) SP=4035d048 RP=a.008e7360  open_ksam_file+$13c
 NM  f) SP=4035cd50 RP=a.008ea17c  reopen_ksam_file+$13c
 NM 10) SP=4035cb00 RP=a.0097362c  fopen_nm+$2a4
 NM 11) SP=4035ca60 RP=a.004b8ba0  FOPEN+$90
 NM 12) SP=4035c350 RP=a.004b8adc  ?FOPEN+$8
```

In a stack trace, the procedure listed at the top is the most recent procedure called. In the above example, the process called FOPEN which in turn called fopen_nm, reopen_ksam_file, etc. The most recent procedure called by this process is FSETMODE followed by SWITCH'TO'NM. A stack trace is a good picture of what is going on with a process. Note that the stack trace in this example has both native mode and compatability mode code. In order to get a complete trace of all mode switches you must use the command TR,I,D.

For the more advanced dump reader, it may be beneficial to jump to a specific procedure level and dump out parameters that were passed to that procedure. This would be done with the lev and dv commands, but we will not be discussing this here.

The machine_state, information on the current process, and a stack trace will all be valuable in looking at any type of failure. It is especially useful in determining whether a repeat failure may have a similar cause as prior occurrences.

System Aborts

It is unlikely that you will learn to adequately analyze a dump of a system abort. Determining the exact cause usually involves in-depth interpretation of procedure parameters and operating system data structures that requires extensive training. What you can look for, however, is the identification of a process that may be suspect and consistencies in repeat failures. The information discussed above will be the most valuable in determining these consistencies.

System Hangs

In analyzing a system hang, we must be aware that there are two types of hangs. One is a true system hang where all processes are blocked waiting on some resource. There will likely be no process executing in this case. The second type is a process loop or starvation, where one process is consuming all the cpu, locking other processes out.

The best macro for determining which type of hang you have is process_dispatcher. This macro shows all the current dispatcher information including the scheduling queue settings, processes on the dispatch queue, and the state of these processes. If you see a process on the dispatch queue with a state of **RUNNING**, it may be a process that is looping at a high priority. This conclusion would only be drawn if there are several other processes on the dispatch queue in a **READY** state. If no other processes are in the READY state, then the you likely do not have a looping or starvation problem. In the case that you suspect a looping process it is best to use the PIN command to select this process and then trace it to get a picture of what it is doing. Keep in mind that it is the normal state of the system to have a process in the RUNNING state so do not jump to the conclusion that this process is at fault unless you have supporting information.

If process_dispatcher shows no RUNNING process, it is likely that you have a true hang situation. In this case, you will want to look into why processes are waiting, and for what resource. Two common resources that could cause a hang are SIRs and Semaphores. A SIR is a System Internal Resource which provides a way of single threading processes on a particular type of operation. To get a list of any SIRs that are currently locked on the system, execute the rm_format_sirs macro.

Semaphores are also a locking resource but they are for data structures. An operating system data structure will have a semaphore that keeps track of the current owner and a list of pins waiting on the resource. To look at the semaphores currently locked on the system use the rm_semaphore macro.

From both of these macros, you should be able to determine if there is one process holding up other processes. Again, you can use the PIN command to select a suspect process and then trace it to see what it is doing.

All of the information given for looking at hangs can also be used interactively in DEBUG to diagnose live process hangs.

Other Macros

These are only a few of the extremely powerful macros available for analyzing memory dumps. Individuals may find other macros that they find useful in their own situations. You can look at other macros using the macl command. Many macros have a given prefix to group them by general topic. Table-3 lists some of these prefixes and some common macros available under these categories. You can use these prefixes to list the macros in a given category. For example, if you want to get a list of all the file system macros, you would enter the following command :

```
$nmdebug > macl fs_@
```

Macro Prefix

fs_	file system macros
pm_	process management macros
process_	process macros
rm_	rsource management macros
ui_	user interface macros

Table-3

Conclusion

The information given in this paper cannot be a substitute for in-depth dump analysis by a qualified HP engineer. Hopefully, this will give you a little more control in getting general information that is valuable to you as a system manager.

PAPER #3809

POSIX Interface for MPE/iX

Rajesh Lalwani

Hewlett-Packard
19447 Pruneridge Avenue 47UH
Cupertino, CA 95014 USA

Abstract

POSIX is an IEEE standard for a Portable Operating System Interface to support application portability at the source level. It forms a basic layer which is a first step towards more extensive sets of standards such as X/Open. Currently, HP plans to provide Programmatic Interface 1003.1 (POSIX.1) and Shell and Utility Facilities 1003.2 (POSIX.2) on HP 3000, Series 900.

This paper presents an overview of the POSIX implementation on HP 3000, Series 900. It describes how MPE/iX users will benefit from the POSIX features such as hierarchical directory structure and how POSIX and MPE concepts will be integrated so that existing MPE users can use new features using existing commands and some new CI (Command Interpreter CI.PUB.SYS) commands, while keeping existing commands fully backwards compatible. This paper also describes new CI commands and changes to existing commands that will assist the system manager in managing the system in an MPE/POSIX environment. Finally, this paper briefly discusses the future direction of the HP 3000, Series 900 Open platform.

1. INTRODUCTION

With Release 4.5, MPE/iX provides POSIX.1 and POSIX.2 interfaces. POSIX.1 is a programmatic interface for programs written in the C language; whereas, POSIX.2 is the IEEE standard for shell and utility facilities. POSIX application developers such as VABs and VARs will need the POSIX developer kit which includes, among other things, the POSIX.2 shell and the relocatable library (RL) LIBCPX.LIB.SYS (/lib/libc.a for POSIX.2 shell). The Fundamental Operating System (FOS) contains the core of the POSIX.1 functions available, but this RL provides the necessary C language interface. Once the object files have been linked with this RL, end users can run the resulting POSIX application on a HP 3000, Series 900. The POSIX.2 shell is not a part of FOS; it is a separate product available for purchase.

New features such as the hierarchical directory structure, new commands such as NEWDIR, CHDIR, PURGEDIR, DISKUSE, and enhanced CI commands such as LISTFILE are all part of FOS as of Release 4.5. This is the main topic of this paper. Section 2 of this paper introduces the new hierarchical directory structure and how it has been integrated with the existing `file.group.account` structure. Section 3 discusses why an existing MPE user might want to use the new hierarchical directory structure, and describes the new commands NEWDIR, CHDIR, and PURGEDIR that deal with hierarchical directories. It also introduces the new features of the existing LISTFILE command that has been enhanced to work with the hierarchical directories. Section 4 discusses the impact POSIX will have on system management. It explains the user and group databases and the PXUTIL utility. It explains the new syntax of STORE/RESTORE for backing up and restoring files in directories. Finally, the new command DISKUSE for reporting disk space used by directories is introduced.

2. HIERARCHICAL DIRECTORY STRUCTURE

The MPE operating system had been designed primarily for use in a department setting. All the users in a particular department are typically grouped together and placed in one account. Similarly, an account, such as `PAYROLL`, could organize its files in groups such as `JAN92`, `FEB92`, etc. In this model, the directory tree consists of one or more accounts, with each account containing one or more groups. Finally, each group has zero or more files in it. Hence, this `file.group.account` structure can be looked at as a 3-level hierarchical structure. The first level has account entries, the second level group entries, and the third level the actual files. POSIX, however, has an expanded version of MPE hierarchy. The POSIX directory structure has a root directory which is represented by a *slash* (/). The root directory may have files and directories under it. The directories, in turn, have files and directories under them. MPE/iX has integrated the MPE and POSIX concepts. A typical MPE/iX directory could look like:

```
                                      /
                                      |
      +-------------------------------+-------------------------------+
      |                               |                               |
   PAYROLL                         DEVELOP                          users
      |                               |                               |
+-----+-----+              +----------+----------+         +---------+--+
|           |              |                     |         |            |
JAN92     FEB92           MPE                    PX      bruna        jeff
            |                                    |                      |
         +--+--+                    +-------+----+----+-----+        +----+----+
         |     |                    |       |         |     |        |         |
      HOURS  TAXINFO             ledger  payable    data  PRGFILE   bin     addr_bk
                                    |                                |
                                 +--+---+                      +-----+-----+
                                 |      |                      |           |
                              main.c  Makefile               my_ls     down_load
```

Figure 1 - Typical MPE/iX Directory

On MPE/iX, the root directory may contain files, directories, and accounts. For most purposes, accounts and groups are treated as directories but there are differences. Like any MPE system, the accounts can contain groups. The groups can, in turn, contain files or directories. In **Figure** 1 above, three different flavors of the integration of MPE and POSIX concepts can be seen. `PAYROLL` account has not used any of the new features of POSIX. The account `PAYROLL` contains two groups `JAN92` and `FEB92`. The group `FEB92` contains just two files `HOURS` and `TAXINFO`. Notice that the files in the group `FEB92` even have valid MPE names. It seems a person familiar with UNIX has created the directory `users` under the root directory. This person has placed two directories, `bruna` and `jeff` under `users`. The directory `jeff` has a simple file `addr_bk` and a directory `bin` under it. The directory `bin` has two script files `my_ls` and `down_load`.

The person using the `DEVELOP` account has really used both MPE and POSIX concepts. The account `DEVELOP` has two groups `MPE` and `PX` in it. This account is apparently used by an application developer. There are two directories `ledger` and `payable`, probably for keeping source and make files related to the two modules of the accounting package. The whole application, after compilation and linking, has been kept in the file `PRGFILE`. Apparently, this package keeps the data in the file `data`.

Here are the highlights of the MPE/iX directory structure:

- The root directory (represented by a *slash* /) can have accounts, directories, or files under it.
- The accounts have groups. Note that accounts cannot have files or directories directly below them in Release 4.5. This restriction may go away in the future.
- The groups can have files or directories under them.
- All directories can have files or directories under them.
- Only accounts and groups must have valid MPE names. All others objects can use the POSIX character set which consists of `A-Z`, `a-z`, `0-9`, and three special characters `.`, `_`, and `-`. No name can start with a hyphen (`-`).
- All names directly below the root, account, and group directories can be at most 16 characters in length (except account and group names which must be valid MPE names). All other names can be 255 characters long.
- In Release 4.5, only the program files residing in an MPE group and having a valid MPE name can be RUN. This restriction may go away in the future.

Current Working Directory

On MPE/iX, every job and session has an associated *logon group* which is used to qualify an unqualified file name. For POSIX name server (see below), MPE/iX uses two new references - **current working directory** (CWD) and **root directory**. It is important to know that these references are defined for each *process*. Thus, two processes in a single session can have two different CWDs. If a pathname begins with a *slash*, the predecessor of the first filename in the pathname is the root directory of the process. Such names are referred to as **absolute pathnames**. For example, `/users/jeff/bin/my_ls` is an absolute pathname. However, if the pathname does not begin with a *slash*, the predecessor of the first filename of the pathname is the CWD of the process. Such names are referred to as **relative pathnames**. `bin/my_ls` is an example of a relative pathname. A special filename **dot** (`.`) refers to the directory specified by its predecessor in the pathname, and **dot-dot** (`..`) refers to the parent directory of its predecessor in the pathname.

Name Servers

We just saw where the objects such as groups, accounts, files, and directories can lie within the MPE/iX directory structure. Let us now see how we can refer to them. MPE/iX has been designed so that the existing interfaces (such as intrinsics and CI commands) can refer to <u>all</u> the objects in the directory without changing the way they refer to the MPE objects such as files in groups. Also, POSIX interfaces such as POSIX.1 functions and POSIX.2 commands are able to refer to MPE objects such as accounts and groups and files within them. Conceptually, there are three name servers: MPE, MPE-escape, and POSIX.

First let us see how MPE interfaces such as intrinsics and CI commands can refer to the objects in the directory. The familiar MPE objects (e.g., files in groups) can be refered to by using `file`, `file.group` or `file.group.account`. All such names are processed by the MPE name server. All these names must be valid MPE names and the MPE name server upshifts the name. When the filename is fully unqulaified (say, `prgfile`), MPE name server first upshifts the name to

PRGFILE and refers to a file named PRGFILE in the (CWD) which may be different from the logon group. In the second case when the filename is, say prgfile.px, it refers to a file named PRGFILE in the group named PX in the logon account regardless of whether the CWD is same as the logon group or not. Finally, in the third case when the file name is say, prgfile.px.develop, the MPE name server refers to the file PRGFILE in group PX in account DEVELOP regardless of where the CWD is and what the logon account is.

However, if you want to refer to a file, say down_load in **Figure 1**, you can not use MPE name server as the file is not under a group. You must somehow *escape* from the MPE name server to the POSIX name server. Here is how you can do that : if any name begins with a *dot* (.) or *slash* (/), the MPE-escape name server does not interpret the name and passes it on to the POSIX name server. So using the name, /users/jeff/bin/down_load will do the job. Or if you change your CWD to /users/jeff, you could use the name ./bin/down_load. Unlike POSIX name server, as we will see next, you cannot use bin/down_load as the name does not begin with either *dot* or *slash* and hence MPE-escape name server will not escape to the POSIX name server and may interpret the name as the file BIN with lockword DOWN_LOAD and will give error as DOWN_LOAD is not a valid lockword.

As we said earlier, even MPE objects such as accounts, groups, and files under them can be refered to by the POSIX name server. For naming purposes, the accounts and groups can be treated as directories. Hence, the file PRGFILE.PX.DEVELOP can be refered to as /DEVELOP/PX/PRGFILE. But the POSIX name server does not upshift the name, hence you cannot use the name /develop/px/prgfile. For the POSIX name server, names are always in POSIX syntax; you don't have to start the name with *dot* or *slash*. So, if the CWD is /DEVELOP/PX/ledger, a filename main.c will not refer to the file MAIN in group C in the logon account - it will simply refer to the file named "main.c" in the CWD. (Remember that *dot* is a vaild POSIX character.) The POSIX name server is not yet available to the CI commands or any intrinsic (except HPFOPEN). The name must begin with a *dot* or a *slash* in case you want to inform the MPE-escape name server to invoke the POSIX name server.

The name server used by POSIX.1 functions and POSIX.2 commands is the POSIX name server. Most MPE intrinsics and CI commands have been modified to use MPE-escape name server. Thus old applications/scripts will continue to work because valid MPE names can never begin with a *dot* or a *slash* (except the COPY command) and thus MPE-escape name server will use the normal MPE name server. However, the user can escape to the POSIX name server by using the *dot* or *slash* character in the beginning. We will see some examples in the next two sections.

3. USING HIERARCHICAL DIRECTORIES

As mentioned earlier, Release 4.5 will have the new POSIX features such as hierarchical directory structure, new CI commands, and enhanced CI commands as part of FOS. The system manager and users do not have to *activate* POSIX to be able to use these features. So why not use directories? Directories definitely provide a much better way of organizing files. Also, POSIX file names can be longer than 8 characters and can use some special characters as discussed earlier. This section will introduce the new commands NEWDIR, CHDIR, and PURGEDIR and the enhanced LISTFILE command. Several examples illustrate their use.

Creating a Directory

Directories can be created directly below the root, another directory, or a group. Note that in Release 4.5, a directory cannot be created directly below an account. To be able to create a new directory under a node, the user must have CD (Create Directory) access to that node. A new CI command NEWDIR can be used to create a new directory. For example, if a user logged on as follows:

```
:hello ledger,engineer/does.develop/ment,px
```

the directory `ledger` could have been originally created by the following CI command:

```
:newdir /DEVELOP/PX/ledger
```

Notice that the account name and the group name are in upper case because the POSIX name server (which is invoked by the MPE-escape name server because of the leading *slash*) does not upshift the name. An alternate way of doing the same thing is to issue the command as follows:

```
:newdir ./ledger
```

The beginning *dot* is used to escape to the POSIX name server. Since the name does not begin with a *slash*, it is CWD relative and refers to `CWD/./ledger` which is equivalent to `/DEVELOP/PX/./ledger` or `/DEVELOP/PX/ledger`. When the user logs on, the CWD is the same as the logon group. This can be changed as we will see shortly. Let us see what is wrong with the following:

```
:comment example of doing it incorrectly
:newdir ledger
```

The above command looks good initially. But it does not do what we intended to do. Since the name does not begin with *dot* or *slash*, the name server used is MPE name server. Since it always upshifts the name, a directory called `LEDGER` is created in the CWD (which is the logon group). Finally, let us see what is wrong with the following:

```
:comment will give error
:newdir ./a_long_directory_name
```

Normally, POSIX names can be as long as 255 characters, but since we are creating a directory directly below a group, it can have a maximum of 16 characters. Only files and directories directly below a directory can have longer names.

Changing Current Working Directory

If the above user `engineer.develop` wants to work in the `ledger` directory under the group `PX`, the CWD can be changed as follows:

```
:chdir ./ledger
```

The new CHDIR command allows a user to change the CWD of the CI process. Of course, all the children processes created by CI, initially inherit the same CWD. Another way of doing the same is as follows:

```
:chdir /DEVELOP/PX/ledger
```

It is possible to change the CWD to any directory as long as the user has proper access. If the user `engineer.develop` has TD (Traverse Directory) access to the root (/) and `users` directories, the following command will change the CWD to `/users/jeff`.

```
:chdir /users/jeff
```

Purging a Directory

MPE/iX provides a new command PURGEDIR to purge a directory. Let us assume that the user has just logged on as follows:

```
:hello ledger,engineer/does.develop/ment,px
```

The user can purge the directory `payable` as follows:

```
:purgedir ./payable
```

However, the following command will not succeed:

```
:comment will not succeed
:purgedir ./ledger
```

The above command does not succeed because the directory `ledger` is not empty - it has two files `main.c` and `Makefile` under it. If the user really wants to purge the directory `ledger` and everything below it, the following command should be used instead:

```
:purgedir ./ledger;tree
```

Finally, the following command will not succeed because PURGEDIR cannot be used to purge accounts and groups:

```
:comment will not succeed
:purgedir /PAYROLL/JAN92
```

Listing Contents of Directories

LISTFILE is one of the many CI commands that have been enhanced to work with the hierarchical directory structure without losing backwards compatibity. LISTFILE has many new features and we will see them in the following examples. On releases before Release 4.5, we use the following command to list all the files on the system:

```
:comment lists all the files on system (before Release 4.5)
:listfile @.@.@,2
```

But on Release 4.5, this command will only display all the files with valid MPE names in all the groups of all the accounts. This set of files could be all the files on the system, if no directories have been created and all the files have valid MPE names. If you wanted to display all the objects on the system (including files, directories, groups, accounts), do the following:

```
:listfile /,2;tree
```

Since the fileset begins with a *slash*, POSIX name server is used. LISTFILE starts displaying objects starting with the root. The `;tree` option tells the LISTFILE command to display everything below the specified fileset. Actually, in this case it was not necessary to specify the `;tree` option because the trailing *slash* at the end of the fileset indicates "everything below it." To account for possible long file names, format 1 and 2 display file names to the right of all other information; this is different from MPE where the file names are displayed first. If you were interested only in files and not directories, groups, accounts, use LISTFILE as follows:

```
:listfile /,2;seleq=[object=file];tree
```

The SELEQ parameter of LISTFILE has been enhanced with a new keyword OBJECT. Other than FILE, you can specify ACCOUNT, GROUP, or DIR. DIR includes both accounts and groups in addition to hierarchical directories. If you use SELEQ, only those objects are listed that satisfy the selection criterion. As another example of the use of SELEQ parameter, the following command will display all the accounts on the system:

```
:listfile /@,2;seleq=[object=account]
```

The above command will not list the directory `users` even though it matches the pattern /@ of the fileset. This is because it does not pass the selection criterion of SELEQ. Finally, let us see how we could find a file on the system. All we know is that the name contains `_load` in it. The following command will do it:

```
:listfile /,6;name=@_load@;tree
```

The new parameter NAME = `<pattern>` further limits the objects displayed by LISTFILE to those whose names match the `<pattern>`. In the above example, format 6 is used to get the fully qualified name of the file once it is found.

4. MPE/iX SYSTEM MANAGEMENT

The users of MPE/iX definitely have the choice whether or not they want to use the new POSIX features such as hierarchical directories, byte stream files, etc. But system managers will need to be aware of POSIX after Release 4.5. If any of the users on their system are creating directories, system managers need to change their STORE/RESTORE procedures slightly. For this purpose, TurboSTORE/iX has been enhanced to work with hierarchical directories. Also, CI.PUB.SYS has a new command called DISKUSE to find the amount of space used by directories (including groups and accounts).

User and Group Databases and PXUTIL

On POSIX systems, each user is identified by a non-negative number known as a user ID or UID. Also, to facilitate sharing of files among users on the system, POSIX supports grouping of users. Each POSIX user is a member of at least one group. A group is identified by a group ID or GID which is a non-negative number. Each system maintains UIDs and GIDs in two databases called user and group databases respectively.

On MPE/iX, the UIDs and GIDs are stored in two files - HPUID.PUB.SYS and HPGID.PUB.SYS respectively. The management of these databases is invisible to the users, such as during NEWUSER, ALTUSER, PURGEUSER, NEWACCT, and PURGEACCT commands. These databases are automatically created when POSIX is initially installed. However, later if for some reason, these databases are corrupted or are not synchronized with the information in the directory, they can be repaired by using a utility PXUTIL. This utility can be used only by a user with SM capability.

Storing/Restoring Files in Directories

Let us assume that some users on the MPE/iX system have started creating hierarchical directories as shown in **Figure 1** above. What will happen if the files are backed up using the following command:

```
:comment will not store all files on Release 4.5
:store @.@.@;;show;directory
```

Like the LISTFILE command above, this STORE command only backs up those files that have valid MPE names and are in MPE groups. This means that only the following files will be stored : `HOURS`, `TAXINFO`, and `PRGFILE`. Of course, it will store the directory structure because of the `;directory` option. If you wanted to store all the objects on the system, you should use the following command instead:

```
:store /;;show;directory;tree
```

STORE/RESTORE uses the POSIX name server because of the leading *slash*. The `;tree` options specifies that everything below the fileset should be stored. You can use the minus operator (-) to indicate the objects that you do not want to store. But since - is a legal POSIX character, minus operator must be surrounded by blanks. As an example, the following command will store everything on the system *except* the directory `jeff` and everything below it:

```
:store / - /users/jeff/;;show;directory
```

In contrast to the above, the following command will store everything except only the directory `jeff`. Even the files `addr_bk`, `my_ls`, `down_load` and directory `bin` are stored because in the command, `jeff` does not end in a *slash*:

```
:store / - /users/jeff;;show;directory
```

Finally, the following command will restore everything except the files in the directory `bin`. The directory `bin` , however, will be restored:

```
:restore ; / - /users/jeff/bin/@;show;create
```

Reporting Disk Space Usage

Release 4.5 provides a new CI command DISKUSE to report the disk space usage of directories, groups, and accounts. For example, if you wanted to see the total disk space used by the entire directory structure, issue the following command:

```
:diskuse /
```

The above command will show the total space used by the root directory and everything below it. It breaks it up further to show the disk space used by each directory node. If you did not want to see the disk space used by each directory and wanted to see just the grand total, use the following command:

```
:diskuse /;notree
```

Finally, if you just wanted to see the disk space used by all the groups in the account `DEVELOP`, and their total (which will not include the disk space used by the account node `DEVELOP` itself), issue the following command:

```
:diskuse /DEVELOP/@;tree
```

5. CONCLUDING REMARKS

This paper described the new features such as hierarchical directories available on MPE/iX and how they can be used via the existing interfaces such as intrinsics and CI commands. It also described how system managers will be affected and the tools that will help them manage the MPE/iX system.

HP is committed to supporting industry standards on the HP 3000. In the future, these efforts will continue. In the short term, some enhancements will be made to allow, for example, programs to run from within hierarchical directories. In the long term, effort will be made to implement all the POSIX.1 functions, achieve X/Open branding such as XPG4. Other POSIX standards are evolving in the areas of real-time extensions, security, and transaction processing. The HP 3000 strategy is to adopt the standards that bring greatest value to commercial OLTP customers.

6. ACKNOWLEDGMENTS

I sincerely thank my colleagues Diane Bassett, Anil Sharma, Thomas Shem, and Jeff Vance who reviewed the paper and provided valuable feedback. I also thank my managers Bruna Byrne and Ann Stein for their support (and the LaserJet).

7. TRADEMARKS

8. DISCLAIMER

9. ABOUT THE AUTHOR

Rajesh Lalwani joined Hewlett-Packard in 1988 as a software development engineer in the MPE Lab of Commercial Systems Division. Currently, he is a member of a team providing POSIX interface on the HP 3000 system.

Paper 3810

ALLBASE/Turbo CONNECT: The Write Stuff

RJ Diepeveen
Hewlett-Packard Company
Commercial Systems Division
Cupertino, California

Feel free to congratulate me. I've recently been promoted to Vice President of Technical Systems. How I got here is an interesting story. And I can't say I did it alone. There's an entire team of people that helped, and one visionary vendor. Let me tell you my tale.

It was a Friday afternoon and I was looking forward to a weekend on my boat. The phone rang, the internal ring, and, foolishly, I answered it. "MIS, this is Steve," I said already thinking my weekend shot.

"Steve, glad I caught you. This is James, in Marketing. You know, the new Marketing Director." And indeed I did know. James was this hotshot marketing type that my company stole from one of the high tech companies out in silicon valley.

"Yes, James, are you getting settled in to the new job?"

"Oh, yes, thanks," he replied. "Listen Steve, I've reviewed the marketing systems installed here and find several critical functions missing. I've identified a package which will address the missing areas, but it uses relational database technology. Is a relational database product on the system now?"

The answer to that question was not as simple as yes or no. Perhaps some background is appropriate. The system we have is an HP 3000 Series 980/400 which is a great box. HP 3000 systems have been the computer mainstay at this company for over ten years. And in that time my department has acquired or written TurboIMAGE based applications for everything. At the time we upgraded from our MPE V systems to the PA RISC system, ALLBASE/SQL was purchased. Even though it is on the system, aside from a little testing to see how it works, sadly, we have never implemented any applications using it as the data management system. So, I quess, in answer to his question, yes, we do have it, but we don't have a great deal of experience using it. Oh, by the way, at this point I was the MIS Manager.

"Well, James, we do have ALLBASE/SQL on the system," I answered trying to make it sound like it wasn't a big deal.

"That's great," he replied enthusiastically. "We should be able to install this package in no time, and the cost to implement is reduced because this package runs on ALLBASE/SQL. Steve, I've got a demo package from the vendor. What say we load it up and take a quick look?"

Oh, the naive amongst us. Here I have a department full of TurboIMAGE gurus and this guy thinks we can just slam this relational database application in before leaving for the weekend. No surprises from this guy yet. "James, I appreciate your enthusiasm and I don't mean to dampen it, but why don't you first explain to me what this application does and the benefits you expect to achieve." Dampen, I did not, serious dry spell was more like it.

Speaking faster than any human should, but maintaining his diction and my attention, James proceeded to explain that this application was a sales forecasting tool which used sales projections and industry trends against actual production plans to determine future order potential. The relational access was essential to data modeling and the resultant forecasts. The benefits were obvious, improved production planning and revenue prediction through more accurate order forecasts. The part that bothered me was the part about using actual production plans. Production plans are stored in TurboIMAGE databases.

I did spend the weekend on the boat, but the marketing system documentation entertained me rather than the sails. What I discovered is that most of the information which this application relied upon already existed in applications on our system, but buried in TurboIMAGE. To avoid redundancy and to maintain current, accurate information for the marketing system, the new relational application needed to access the original sources of the data, it needed to access TurboIMAGE.

This was less of a problem then I thought. HP offers ALLBASE/Turbo CONNECT, the SQL to TurboIMAGE bridge. This product basically allows TurboIMAGE information to be viewed as relational tables accessible by SQL commands. This would address the modeling and forecasting aspects of the marketing application. But there was a feature that James has overlooked. His new marketing system could supply forecasted order information to production scheduling for inclusion in the MRP runs. This marketing application, using ALLBASE/Turbo CONNECT, could write orders into the production plan. Thank you HP for ALLBASE/Turbo CONNECT.

"James," I said, smugly, "this appears to be a pretty good system. I've reviewed all the information you gave me and have written up an implementation plan which I have forwarded to you in HP DeskManager."

"Yea, Steve, I just read it. But where is this legacy system interfacing capability coming from?" he asked.

"It's this slick product called ALLBASE/Turbo CONNECT which lets SQL use our TurboIMAGE data as relational tables. As you can see from my report, the benefits of your new application will start saving the company many, many thousands of dollars in labor and material costs every month."

"Golly!" was all he said.

That, however, was just the beginning. The first thing we did was tear into the ALLBASE/SQL literature. The marketing application was installed with its own DBE. That's "Database Environment" if you're new to this relational stuff like I am. To get ALLBASE/Turbo

CONNECT (affectionately referred to as ATC) to look at TurboIMAGE relationally, all you have to do is run this utility called ATCUTIL and ATTACH the TurboIMAGE database to the DBE. It was so simple, even I could do it. And I'm a manager!

I couldn't wait to test it. Using Interactive SQL, ISQL, I connected to the DBE and used a SELECT command against a data set. There, right before my eyes, appeared row after row of TurboIMAGE entries. I was as excited, no, more excited than James was when he first called me that Friday afternoon. But it wasn't his marketing system that fueled my excitement. What I was thinking about was Client/Server Computing. HP is a leader in this area, and I had been looking for a justifiable and cost effective way to put Client/Server technology to work here at my company.

Perhaps I've lost you, so let me explain. Several software vendors offer API products. API is Application Programming Interface. HP offers one called ALLBASE/SQL PC API. These products allow you to write Client based applications which access a Server where the data is located. Of course, the data is assumed to be in ALLBASE/SQL. But with ALLBASE/Turbo CONNECT this data can be in TurboIMAGE. Client/Server with TurboIMAGE!!! When that marketing guy called and I thought I was dampening his enthusiasm, little did I realize that I was actually irrigating incredible new ideas for computing in this company.

This whole story is starting to get a little fragmented in my own excitement. But there are really two main points to be taken. First, the marketing system became an overwhelming success because of the integration we were able to achieve with ALLBASE/Turbo CONNECT. Second, this same product, ATC, would offer significant advantages to our company through distributed processing and client/server technology. This will allow us to put PC and HP-UX workstations in our employees' hands to improve their productivity while maintaining controlled growth of our HP 3000 server system. Let me tell you what we did.

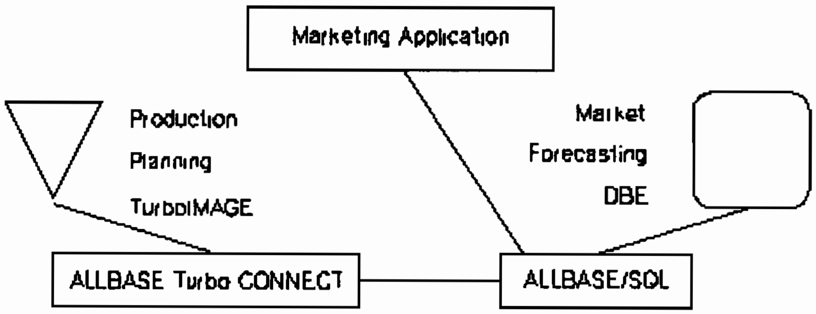
The marketing system became our pilot project. The standard package included interfaces for importing data from external sources. Our plan was a bit more radical. Since source code was included, we changed the table lookup from the imported data in the ALLBASE/SQL table to the original source of data in the TurboIMAGE data set. This gave the marketing system accurate and timely data which was not redundant. This, of course, was done via ALLBASE/Turbo CONNECT. This ATC conduit also became the path for updating production planning information in the production systems. An important thing to keep in mind here is that all of our original TurboIMAGE applications are continuing to operate unchanged. Marketing system users have concurrency with all the standard MRP application users.

The second ATC project we embarked upon again involved the marketing application and was our first Client/Server application. All of our sales reps have either a PC in their home or a portable PC to carry around with them to their customer sites. Using ALLBASE/SQL PC API, we designed a Client/Server application which links the sales rep with the marketing application and the order processing system which is in TurboIMAGE. Here, again, ATC was used for this access. The sales reps can now input orders online, quote delivery dates, update their sales forecasts and inquire into market trends. All information either stored in ALLBASES/SQL or TurboIMAGE databases.

The design options open to us now are incredible due to so many vendors creating open system tools. And we pick the best tool for the job. If speed is of utmost importance, intrinsic access to TurboIMAGE is used. If new or revised functionality is desired, SQL application development tools are used to reduce development time and ATC is used to supply the access. Another benefit is the new relational look we have of our TurboIMAGE data that ATC provides. Using any of the many ad hoc inquiry tools, TurboIMAGE data can be joined and selected in a myriad of ways to make even the most analytical manager happy. And inquiry can join TurboIMAGE tables with ALLBASE/SQL tables.

Let me show you some graphic examples of what we have done and are planning to do with ALLBASE/Turbo CONNECT in our business. In figure 1 you see the marketing application simplified as we implemented it.

Figure 1

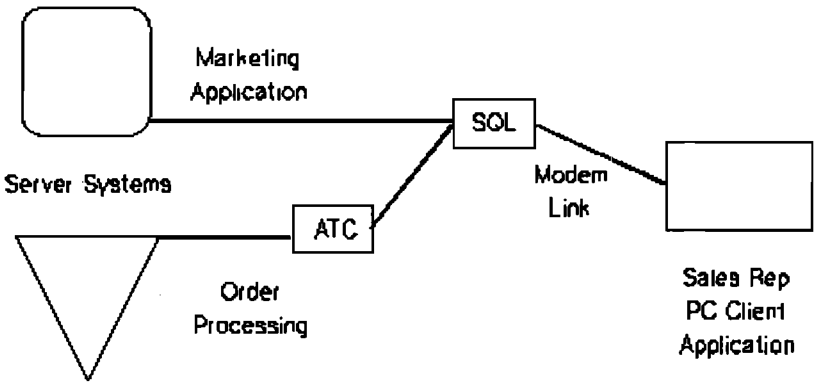


As you can see, when the marketing application needs the production plans it accesses the information as a relational table residing in the TurboIMAGE database. When those production plans need updating, ATC is again the route used to update those plans.

The implementation for the sales rep system looks like that in figure 2. In this case, the Order Processing database, a TurboIMAGE database, is accessed along with the relational DBE of the Marketing system. Actually, the way ATC works, there need not even be a relational database defined within the DBE. It could just contain the interpreted TurboIMAGE database. That is not the case here, however. The API application can access both the Marketing and Order Processing databases. This application, when complete, will also be used by our internal order processing staff here at the corporate offices. Their access will be as client PCs using the HP 3000 server host.

As mentioned above, we are not limited to having relational tables in the DBE. TurboIMAGE tables alone can reside in a DBE. Right now many of our PCs are using a terminal emulator to access the HP 3000 over the LAN. The applications being run are COBOL programs using VPLUS as the user interface. This means that the HP 3000 is supplying all the processing power required to perform this application. Using PC Client/Server tools, we can rewrite these applications so that the PC is the processing power for the user interface and for processing, including local data storage. By moving this processing to the client PCs, and making the HP 3000 the Server, processing power on the HP 3000 is reserved for either adding more users or for adding new functionality, but without having to upgrade the entire system. From the point of view of the Client applications, the data sources are ALLBASE relational tables, but we know better.

Figure 2



The engineering department recently came to me with a request to access current product designs and bills of material. This is stored in the MRP TurboIMAGE based application. ATC can help here, but the twist is that the engineering department is using HP 9000 workstations. In this case, the ALLBASE linkage is handled with ALLBASE/Net over the LAN. ALLBASE/Net sees the MRP system as just another set of DBE tables, rather than the TurboIMAGE data sets that they really are.

By now you can see that Hewlett-Packard has developed an impressive array of SQL related products which allow you to do some incredible things. ALLBASE/Turbo CONNECT gives you SQL access to your TurboIMAGE legacy systems. ALLBASE/SQL provides you with direct access to the relational tables. ALLBASE/Net gives to network access to ALLBASE/SQL and TurboIMAGE via ALLBASE/Turbo CONNECT. ALLBASE/PC API will give you Client/Server processing capability.

Project after project was begun and successfully completed. Department after department started making better decisions, better products, and offering better service. Our market share soared, as did our profits. Following each project, reviews were done to determine if improvements were noticed, and in all cases the answer was that, yes, improvements were seen with regard to information necessary to perform the required functions.

The company president announced my promotion in the auditorium in front of everyone that had come to collect their profit sharing check. I don't know if your company can benefit as much from implementing these products and this technology, and I don't know if you'll make Vice President, but you never know. So why don't you look into it.

PAPER #3811

Introduction to MPE/iX Architecture

Thomas Shem

Hewlett-Packard
19447 Pruneridge Ave 47UH
Cupertino, CA 95014

Abstract:

The MPE operating system has evolved over the past twenty years to accommodate the needs of the user community by continually providing new features and new technologies. With the introduction of MPE/iX, MPE is continuing this evolution by incorporating many of the features found in Open Systems.

This paper provides an introduction to the architectural evolution that has taken place from MPE XL to MPE/iX. It describes the significant enhancements which were made to the directory, to process management, and to the file system. It illustrates the evolution of the system's perception of a user and the user's environment. It also details the enhancements to system security policies necessary to accommodate this evolution.

This paper is intended for intermediate to advanced audiences. The audience should have a working knowledge of the MPE operating system in general (MPE V or MPE XL). The approach this paper takes is to compare the pre-MPE/iX architectures to the MPE/iX architecture.

INTRODUCTION:

In the light of recent industry trends, the MPE operating system is experiencing a new renaissance. This renaissance is occuring because of HP's commitment to make the HP 3000 an Open system. As part of HP's open system program, MPE has incorporated parts of the IEEE 1003 Standard, otherwise known as the Portable Operating System Interface standard (POSIX). This standard incorporates many of the more common functions and interfaces found on many UN*X systems.

The MPE and UN*X operating systems are very different from each other. The UN*X philosophy is to try to keep all of its parts as small and modular as possible. This provides programmers with a large set of basic building blocks which can be used to construct applications. The downside of this philosophy is that UN*X users must be very knowledgeable about the UN*X system. This is demonstrated by UN*X's use of very terse and sometimes cryptic command names (awk, yacc, etc...), as well as its lack of detailed error messages or recovery instructions. Often on UN*X, it's a case of "let the

A GUIDED TOUR THROUGH MPE/iX:

In this article, we will take a brief guided tour through the MPE operating system in order to explore the changes introduced in MPE/iX. MPE users will find some of the POSIX concepts to be very foreign. Just like a visit to a foreign country, the traveler may become very uncomfortable with the local behaviors and customs. With time, the foreign traveler finds that they are enriched by their experiences.

Our expedition will begin by examining the evolution of how MPE/iX defines a user and the new enhancements to the user's working environment. We will search into the depths of the new system directory and file system, looking for the most significant changes. We will learn new ways of communicating from process to process by way of the process management changes. Lastly, we will weave our way through the treacherous waters of system security, hopefully to find our way home again.

So, let's pack our bags and collect our passports. Our journey is about to begin.

Travel Note #1: Identification of The User

As we attempt to enter MPE/iX, we are stopped at the border by the local security agents. We greet them with a hearty HELLO, and present our id. After careful scrutiny, we are promptly welcomed by the locals with the usual banner and fanfare.

Long time MPE users know that MPE was originally designed for use in a department setting. Users of the system were typically grouped together by departments which were associated to an MPE account. The user would thereafter be identified by their user.account string.

With the binding between the user and account established, the user's possessions (files) were stored within the account under MPE groups. The owner of those files were identified by placing the user's name within the file label.

The consequences of this architecture are that both the user and the files created by that user are tightly bound to the account. This means that both the user and the files can not travel outside of the logical account boundary. The object (user or files) can be duplicated, but can not be physically moved.

The POSIX/UN*X identity model is not like MPE's identity model. POSIX/UN*X identifies each user by a unique user ID number (UID). Users are arranged into groups of users and identified by a group ID number (GID). Owners of files on POSIX/UN*X are identified by placing both the UID and the GID within the file label.

Unlike MPE, the data associated with the user is stored in separate user and group databases. This means that on UN*X, users are not bound to any particular directory structure. So, both user objects and file objects are free to move around within the directory structure if system security allow it.

In MPE/iX, the POSIX/UN*X identity model is incorporated into MPE. Users are still identified by their user.account string, but will also have a unique user ID number attached to it. Similarly, MPE Accounts will have associated to them POSIX group ID numbers. This one-to-one mapping between user string and UID number and between account string and GID number provides a good compromise for merging the two concepts together.

Ownership of files is similarly affected by the new identity model. The owner is still identified through the creator field of the file label. However, the enhanced creator field now contains the entire user.account string. This preserves the one-to-one mapping from user to UID.

Travel Note #2: Getting around in the User Environment:

After our entry into MPE/iX, our guide showed us around via MPE's primary vehicle for navigating the system, the MPE/iX command interpreter. It's an updated model of the one we were already used to. Equipped with lots of bells and whistles, it provides excellent transportation for maneuvering through the system functions. The snazzy new controls are a little different from the ones we're used to but with a little practice, we're sure we can manage.

Our guide also showed us the newer foreign model. Very lightweight with none of the frills we're used to, the shell looks like nothing we're familiar with. It costs a little more, but enthusiasts already familiar with the foreign model should have no trouble using it. All the controls are where the enthusiast would expect them to be...

The MPE Command Interpreter:

The MPE Command Interpreter (CI) has been the mainstay user interface for the HP 3000 since its inception. From it, the user can perform a wide variety of functions, ranging from managing the system to program development to application execution. Users gain access to this functionality by using the many built in commands, by using scripts built with the CI's command language, or by executing programs.

In MPE/iX, the MPE CI was extended to support the new POSIX features. Some of the existing commands like NEWUSER, LISTUSER, LISTFILE, ALTSEC, and others, were modified to accommodate the additional functionality within the scope of the existing command. For example, the NEWACCT command contains two new parameters, UID= and GID=, to allow the system manager to assign explicit UID and GID numbers to the manager user and to the new account. Command extension was done whenever possible to maintain continuity for long-time MPE users. Twenty-three CI commands and four utilities have been modified (Please see end of article for complete list).

Whenever new functionality was introduced which didn't match any existing MPE command, a new command or utility was created. In actuality, only four new CI commands (CHDIR, NEWDIR, DISKUSE, and PURGEDIR) and one utility needed to be created. MPE users can probably already guess what the "DIR" commands do based upon their similarity to existing MPE commands. They provide functionality similar to the CHGROUP, NEWGROUP, and PURGEGROUP commands except they apply to the hierarchical directories. The function of the DISKUSE command is similar to the REPORT command in that it reports disc space usage in the hierarchical directory. A utility PXUTIL.PUB.SYS was added to initialize and to provide error recovery for the user or group databases.

The MPE/iX Shell:

An alternative user interface and user environment which correspond to the POSIX environment are defined in the IEEE 1003.2 and 1003.2a draft standard. It describes a shell very similar to the Korn shell available on most UN*X system. The standard defines not only the shell environment but also many commonly used command and utilities.

On MPE/iX, the MPE/iX Shell and Utilities product (hereafter referred to as the shell) will provide the interface which most POSIX developers will expect. The UN*X-like environment will allow users to port their Open applications to MPE with a minimal amount of retraining. Users will find a large number of utilities that they are already familiar with accessible from within the shell.

The shell provides a very basic user environment. Basic building blocks like regular expression evaluation, environment variables, I/O redirection and piping are available through the shell (all but piping have like counterparts in the MPE CI). With these building blocks, users may construct more complicated scripts (command files on MPE).

The shell provides very few built in commands. Unlike the MPE CI, most functions are accomplished through utility programs. This is motivated by UN*X's philosophy for keeping everything small and modular.

With the MPE/iX shell, some 100+ utilities are made available as programs. Long time users of the MPE CI should note that they will be able to execute these utilities from the MPE CI if the HPPATH variable is set up to include the location of the shell programs. However, this will not work for shell utilitities which rely on a function specific to the shell. An alternative way to accomplish this task is to create command files or UDC's which execute the shell, passing into it the shell syntax for a function.

Users trying to use these utilities should beware because the names for the utilities are very terse, or may not be representative of their function. Also, the utilities behave in the same manner as on UN*X-like systems (i.e. improper use of the utilities may result in unpredictable results; very terse error messages or sometimes no error message at all will be displayed if a utility fails).

Travel Log #3: Exploring the depths of the MPE directory

While out sightseeing, we got a little lost. We could have sworn that the last turn ended in a dead end. We are amazed to find that it actually opens up into many more twists and turns than we expect. Just goes to show you, never travel without a good map...

The traditional MPE directory was very simple in design. It consists of a collection of MPE account nodes, MPE group nodes, and file nodes. Files are grouped under MPE groups and MPE groups are bound to MPE Accounts, making up a three level directory tree.

To mimic the department setting, the MPE V directory is fashioned so each department can have a representation as an MPE account. Working groups under that department can be logically assigned to MPE groups. Work done by each group can be kept in the form of files within the MPE groups. Users of each working group are placed into an MPE group during login and are expected to re-login to another MPE group when their working group was changed.

With MPE XL, this representation was modified slightly. Instead of working groups, MPE groups became representations for tasks. Work for each task is then contained in an MPE group. Users could switch from task to task (via the CHGROUP command) as needed.

These representations are both an advantage and a disadvantage. If a user's actual working environment can be modeled to fit MPE's artificial segmentation, the user benefits. If the working environment does not match MPE's representation, the user experiences unnecessary restrictions.

With MPE/iX, the artificial environment of the MPE directory structure is becoming more versatile. Users familiar with the MPE directory structure are in for a pleasant surprise. Users will be able to define and partition their own working space as they see fit. This is done through new functions provided through the CI, the shell, and the POSIX.1 functions.

The directory node concept is being introduced to MPE. A directory node is similar to an MPE group in that it is a place to hold a group of files. Additionally, directory nodes may be created within other directories, creating a hierarchy of directories. This gives users the ability to partition and segment their files into logical and physical groupings which better correspond to how they work.

The familiar MPE account and MPE group structures will be represented as specialized directory nodes within the MPE directory. They will continue to have associated with them attributes like, file access restrictions, passwords, capabilities and other items.

MPE groups will continue to be bound to MPE accounts and MPE accounts will be bound to another specialized directory node called root. Users will be able to create directories and files within MPE groups, root, or other directories, however not in MPE accounts.

MPE users logging into the system will continue to log into their HOME group within their MPE account. This group becomes their current working directory (CWD). Users can still change their location to another group via the CHGROUP command. This also changes their CWD to the new group. Similarly, users will be able to change their location to any directory, for which they are allowed access, via the CHDIR command. This changes the users current working area or CWD to the new directory.

Travel Note #4: Visiting the File System

Finally figured out how to get around, it wasn't so difficult after all. We just have to remember that in the older part of MPE/iX, many of the structures have two addresses.

Our guide showed us a snazzy new structure. The architecture's much more streamlined than the older one, but when viewed from a different angle, appears to be quite contemporary. We were quite surprised to learn that the owner wasn't who we expected it to be. We were also amazed to hear that it had just been moved from the other side of town. It must have been quite some renovation project.

File naming:

The traditional MPE file naming syntax was a good representation of the three level model used by the MPE directory. The syntax described the file by positioning each part, the file name, group name, and account name, in relation to the other parts of the syntax (i.e. 'file.group.account', file always came first, group always came second, account always last).

Unfortunately, the MPE file naming representation does not work very well in the hierarchical directory model. To this end, an alternate hierarchical file system (HFS) file naming syntax has been introduced. This syntax is similar to the file naming conventions used on UN*X and DOS systems.

The HFS syntax consists of two parts, a path descriptor part and a file name part. The syntax describes files by first describing the path or location. For example, the HFS filename '/SYS/PUB/CI' is the same as the MPE filename 'CI.PUB.SYS'. An explanation of the syntax is as follows (NOTE: the '/' character is interpreted to mean directory):

- start at root	- '/'
- go to the SYS directory	- 'SYS/'
- go to the PUB directory	- 'PUB/'
- get the file	- 'CI'

In this manner, all files in the hierarchical directory can be described. Users should note that the HFS syntax is an alternative to the old MPE syntax. The MPE syntax will continue to work for files located within MPE groups.

The file naming resolution is accomplished via the file name server. The MPE name server follows the rules for the MPE file name syntax, upshift name, validate acceptable character set [A-Z, 0-9, ., /], and validate acceptable component length. The new HFS name server provides the same function for the HFS file name, validate acceptable character set [a-z, A-Z, 0-9, ., _, -, /]. and validate acceptable component length. MPE/iX determines which name server to invoke based on the leading character of the filename. In the MPE CI, a '.' or '/' means invoke the HFS name server, otherwise default to the MPE name server. The shell and POSIX.1 interfaces only invoke the HFS name server.

Byte Stream Files:

Traditionally, data within files on the MPE system are organized into records of fixed or variable size. Records are delimited by a logical 'end of record' marker. MPE uses the record concept to speed up I/O when accesses to files is done.

On UN*X systems, data within a file is not organized into records. Users who desire the data to be arranged in a specific format are expected to do so themselves. The UN*X operating system does not support the wide variety of file types that MPE does.

On MPE/iX, a new file type, byte stream, provides the same file structure as found on POSIX/UN*X systems. Byte stream files consist of a continuous 'stream' of data without separation. In MPE terms, the file consists of one byte records. Users accessing a byte stream file will retrieve the data in the same manner as they are accustomed to on UN*X systems, as a series of bytes.

This is done through the new byte stream type manager. In MPE, there is a type manager for every supported file type. They manage and maintain the rules that the file system needs to follow when operating on a file.

Byte Stream Type Emulator:

During the early development period of MPE/iX, the development team was concerned about how traditional MPE users would access a byte stream file. Accesses to a byte stream file is different from a record oriented file. If a user were to display a byte stream file in a traditional MPE manner, they would see one byte per display line. This was felt to be unacceptable and unusable. All existing applications which acted on files would also experience this strange behavior.

To aid in the backward compatibility of MPE, a byte stream emulator was added. It essentially represents byte stream data as a variable length record file. Traditional MPE applications accessing the byte stream file will appear to be accessing a variable length record file. This will allow most applications to work with the byte stream file type.

Changing File Owners & Moving File locations:

As mentioned earlier, the file creator field in the file label of a file on MPE/iX systems contains the user.account identifier of the creator. In part, this change facilitated two additional file system features as defined in POSIX: the ability to change the owners of a file and the ability to move the file to anywhere in the directory (even across MPE account boundaries).

On-pre-MPE/iX systems, the creator is represented by an eight character user name. This user name could not be unique since the same user name could be duplicated for every account on the system. Therefore, if a file A.B was created by user TOM and file B.A was created by user TOM, there was no way of distinguishing the difference between the user TOM.A and TOM.B. The same problem exists if the file A.B is moved into the A account, MPE can not distinguish the two users. By identifying the user by the full user and account identifier, MPE now has a way to uniquely define the ownership of a file.

In the past, the only way to accomplish these two tasks was to first be logged on as the destination creator and second to copy the file. With MPE/iX, the creator of a file can now give up his ownership of the file or move the file to anywhere in the directory.

Travel Note #5: Communicating with the locals

Talked to some of the locals today. Boy, do they talk funny. An ancient tribe appeared to do so through a complicated series of signals. Another more culturally advanced group spoke with a more familiar dialect, although they had a strange customs of sharing and leaving messages around.

In MPE/iX, many new mechanisms have been added to facilitate interprocess communication (IPC). Two types of standards have been added which expand the IPC capabilities of MPE, POSIX signals and AT&T System V IPC. The POSIX standard supports a series of SIGNALS or traps on events. AT&T System V IPC standard supports three mechanisms: shared memory, message queues, and semaphores.

Shared Memory:

The shared memory facility allows processes to communicate via a common memory area mapped into the process address space. If a process modifies the contents of this area, it is immediately available to other processes. The most common usage of this facility is to allocate application global data needed by all the processes. Other examples include the allocation of global data buffers when two or more processes are communicating with each other. The data does not have to be copied from process A to process B, it is immediately available to process B.

Message Queues:

Message queues are another form of process to process communication. Data sent from a process to a message queue is copied into a data buffer which is then linked into the list of data buffers for that queue in a FIFO fashion. Processes can read and write to message queues in any arbitrary order.

Semaphores:

The last mechanism offered by System V IPC is the semaphore facility. Semaphores are a form of IPC which do not exchange data. Instead, they are used to synchronize operations of multiple processes competing for the same resource. The most common usage of the semaphore facility is to synchronize access to the shared memory areas.

Signals:

Signals notify processes of events, such as hardware exceptions, that occur on the system. The purpose of signals in MPE/iX is similar to setting traps on MPE XL through intrinsic calls, such as XCONTRAP and XSYSTRAP. The traps typically intercept a specific event and invoke a trap handler. Signals can also be used in lieu of semaphores to control processes.

There are nineteen types of signals defined in POSIX.1. All have been incorporated into MPE/iX. Examples of these signals are: child process has terminated (SIGCHLD), stop/suspend process (SIGSTOP), and time out has occured (SIGALRM). Additionally, POSIX signals has reserved two events for definition by a user application (SIGUSR1 &SIGUSR2).

Travel Note #6: Security

Got into a little trouble with the law today. We encountered an entirely new organization. Our guide informed us that the policies and procedures we needed to follow were very similar to the ones we had at home. We'll have to bone up on them when we get back home.

Evolving the security model used by MPE proved to be one of the more difficult tasks. This was due in part to the way the IEEE 1003.1 standard and 1003.6 draft standard were laid out. Within the POSIX standards, inconsistencies are inherent from one standard to the next.

The MPE implementation of the POSIX security standards provides a compromise between POSIX and MPE. At the same time it does not sacrifice any of the reliability which users of MPE have grown to depend on. The new security features are integrated in a manner that allows no security breaches to occur. All existing MPE security features will continue to work as in pre-MPE/iX systems.

The security model on MPE was expanded to accommodate the new POSIX features in many areas. The three major areas where the MPE security model was most effected were user identification, directory access, and file access.

User Identification:

As we have already seen the model used to represent a user has been enhanced to include two alternative identities, a UID and a GID. These alternative identities have been added to all file and process structures to facilitate identification of object ownership.

In addition to the UID and GID identities, UN*X systems segment users into three categories: owners, group (NOT MPE groups) members, and other. An owner is any user whose UID matches the object's UID. A group member is any user whose GID matches the object's GID. All other members are the set of all other users. For the purposes of assigning access rights, logical representations of these three groupings as expressed through MPE as follows: 'user.account' or '$OWNER' for owner; '@.account' or '$GROUP' for group members; and '@.@' for other.

Directory Access:

The addition of the new directory objects caused the creation of a security model for them. The access attributes associated with directories are creating directories (CD), deleting directories (DD), traversing through directories (TD), and reading from directories (RD).

When a user attempts to create a directory, access for CD in the parent directory is checked. CD grants the right to make a directory within another directory. Purging or deleting of a directory or file requires that the user have DD access to it. When a user attempts to CHDIR to another directory, TD access to the destination directory as well as all intervening directories is required. Lastly, to view the contents of a directory, RD access is required.

Upon creation of a directory, the default security is created which only allows users with appropriate access (the creator, SM users, and in some cases, AM users) to access or modify the access to the new directory.

File Access:

From an MPE standpoint, all file access security issues created by the MPE/POSIX merge were able to be accommodated through existing MPE file access security models. Access to files is determined by first clearing access to the directory nodes above it. In a file called '/SYS/PUB/dir1/dir2/file', access to the SYS, PUB, dir1 and dir2 directories first need to be established. If accessors fail to gain access to any of the directories, they fail to get access to the file. After the accessors have been cleared through the directories, the user is then checked against the type of access he wishes to perform on the file. If access has been allowed, the user operation succeeds.

Access Mechanism:

The mechanism which MPE/iX uses to represent the access attributes are known as Access Control Definitions (ACDs). ACDs are lists which explicitly grant access to users on an individual or grouping basis. ACDs are assigned to objects via the ALTSEC command or the HPACDPUT intrinsic. Access is granted or denied via the acdpair entries of the ACD. The userspec portion of an acdpair can define individual users (user.acct or $OWNER), groups of users (@.acct, or $GROUP), or all users (@.@). The modes portion of the acdpair defines the type of access the user wishes to grant to other users. For files, access types are read(r), write(w), lock(l), append(a), execute(x), no access(NONE), or copy and read the ACD permission (RACD). For directories, access types are create directory entries (CD), delete directory entries (DD), read directory entries(RD), traverse through the directory (TD), no access (NONE), or copy and read the ACD permission (RACD).

Users of the MPE V/V-delta-4 forward or MPE XL 3.0 version of ACD mechanisms will note that the enhancements involved are the addition of the directory access types, and the inclusion of new system defined user types ($OWNER, $GROUP, etc).

One restriction introduced in MPE/iX involves the use of the lockword mechanism. With MPE/iX, lockwords will be restricted to the traditional MPE directory structure (i.e. only files under MPE groups). The reason for this is that lockwords are viewed as an MPE proprietary mechanism, not covered in the Open system standards. Therefore they should not be propagated to the hierarchical directory.

Conclusion:

Upon the conclusion of our tour, we noticed that people are really the same everywhere. It's just a matter of different customs and upbringing which tend to make us appear *different. So for us, we discovered that we really did have a lot in common with our foreign cousins and that there wasn't so much that was different between us after all.*

For all practical purposes, many people will view MPE/iX as a merging of two somewhat dissimilar operating systems. In reality, what was done was to extend the existing MPE functionality with the POSIX functionality not already found in MPE.

Viewed another way, MPE is being reborn. The new functionality will give long time users new tools from which to build. It will also attract new customers in part due to its conformance to the open standards.

In terms of MPE vs UN*X vs POSIX, an operating system is an operating system. They exist to manage machine resources and to provide users with a useful set of system functions. MPE/iX is a solid realization of this fact.

Appendix

Summary of MPE commands affected:

ALTSEC	: Enhanced to change access permissions for hierarchical directories
ALTUSER	: UID parameter added
BUILD	: supports byte stream files
COPY	: enhanced to copy files with HFS names
FILE	: supports byte stream files; supports HFS names in the filereference parameter
LISTACCT	: Adds FORMAT parameter to provide a choice of display formats
LISTEQ	: enhanced to display HFS names
LISTFILE	: Enhanced to traverse the hierarchical directory tree
LISTGROUP	: Adds FORMAT parameter to provide a choice of display formats
LISTUSER	: Adds FORMAT parameter to provide a choice of display formats
NEWACCT	: GID and UID parameter added
NEWUSER	: UID parameter added
PRINT	: supports byte stream files and HFS names
PURGE	: supports HFS names
RENAME	: supports HFS names and rename across account boundries
RESTORE	: enhanced to restore hierarchical files and directories
STORE	: enhanced to store hierarchical files and directories
VSTORE	: enhanced to verify hierarchically named files

RAPID DEVELOPMENT OF CLIENT-SERVER APPLICATIONS USING RPC

Paper #3812
Charles Knouse
Information Networks Division
Hewlett-Packard Company
19420 Homestead Road Cupertino, CA 95014
(408) 447-2354

ABSTRACT

There are two major techniques for developing client-server applications: IPC (InterProcess Communication) and RPC (Remote Procedure Call). The IPC technique, as provided by BSD Sockets and NS NetIPC, requires the developer to code most of the network access. The RPC technique, provided by the HP Apollo Network Computing System (NCS) and the Open Systems Foundation Distributed Computing Environment (OSF DCE), provides the developer with a higher level of abstraction and hides most of the networking details. This paper compares the steps and effort needed to develop client-server applications using the IPC and RPC techniques. It shows that the RPC technique can increase the productivity of the application developer.

INTRODUCTION

In a client-server application, one program requests that another program perform some task on its behalf. The first program is the *client*, the second program is the *server*, and the task is the *service*. Usually, the client and server execute on different systems, so the service request from the client to the server (and the server's response back to the client) must be transmitted over a network.

By splitting up the tasks of an application, the client-server approach allows each task to be performed on the best suited platform. For example, a window user interface may be provided by clients running on PCs or low-cost workstations. Other client tasks that do not require a graphical user interface might run on terminals attached to a multi-user system. Servers for computation may execute on high-power workstations or even supercomputers. Database access may be provided by servers on systems that maintain departmental or corporate databases. The platforms chosen for each task may be provided by different vendors and run different operating systems.

Clients may use one or more servers during the execution of a task. For example, a client may request data from the corporate and department servers and send the data to the compute server for a computation. Servers may be used by any number of clients. Sometimes a server may be called upon to service several clients concurrently. During the execution of a request, a server may in turn request a service of another server, becoming a client to that server.

A developer of a client-server application must solve two central problems:

1. How does a client locate a server that offers the desired service?
2. How does the client communicate service requests to the server and the server return responses?

There are two major programming techniques that can be used to solve these problems. These are IPC (Interprocess Communication) and RPC (Remote Procedure Call). We will examine the steps required by each technique to program a client-server application.

INTERPROCESS COMMUNICATION

In general, IPC allows two programs to establish a network connection between them and to exchange data over the connection. Typically the connection uses an underlying network protocol, say the Transport Control Protocol (TCP) of the ARPA protocol suite. Some form of IPC is offered by most network software packages. Table 1 shows some of the products that provide IPC mechanisms on HP computers.

Table 1: IPC Products

IPC Products	BSD Sockets[1,2]	NS NetIPC[3,4]	Message Files[5]	LAN Mgr Named Pipes[6]
Platforms (OS)	MPE/iX HP-UX MS-DOS	MPE/iX MPE/V HP-UX	MPE/iX MPE/V	MPE/iX HP-UX MS-DOS
Protocols	TCP/IP UDP/IP	TCP/IP	TCP/IP (NS RFA)	TCP/IP

Create socket	socket()
	bind()
		ipccreate()	fopen()	DosMake NmPipe()

Connect	connect()	ipcconnect()	fopen()	DosOpen()
Accept connection	accept()	ipcrecvcn()	fopen()	DosConnect NmPipe()

Table 1: IPC Products

IPC Products	BSD Sockets[1,2]	NS NetIPC[3,4]	Message Files[5]	LAN Mgr Named Pipes[6]
Send data	send()	ipcsend()	fwrite()	DosWrite()
Recv data	recv()	ipcrecv()	fread()	DosRead()
Close connection	close() shutdown()	ipcshut down()	fclose()	DosDisconn NmPipe()
Send/recv datagrams	sendto() recvfrom()	N/A	N/A	N/A

The following discussion will use BSD socket calls to illustrate the use of IPC for client-server applications.

The server is usually run before the client. It must first create a *socket* to be used in setting up connections with clients (step 1 in Figure 1). For a client to communicate with the server, it must be able to locate the server's call socket. There are several ways this can be done (which will be examined later), but for now assume the server binds its socket to a *well-known address*. This address is a combination of the network address of the system running the server (the IP address) and an identification of the specific socket on the system (the *endpoint* or TCP port number). After creating the call socket, the server must listen for connection requests from clients.

When the client is run, it creates its own socket with an assigned address. Then it connects its socket to the server's socket, using the server's well known address (step 2). After the server accepts the connection, data can be exchanged between the client and the server.

Next the client must build a *request packet* to be sent over the connection to the server (step 3). This packet holds all the data values needed by the server to perform the request (designated i_1, i_2, ... i_m in Figure 1). These data may exist in a number of variables in the client's address space. They must be copied into a contiguous buffer for transmission over the connection. This action is called *marshalling*. The client then sends the request packet on the connection to the server and waits to receive a response from the server (step 4).

The server receives the request packet from the client over the connection (step 5). It gets the data i_1, i_2, ... i_m out of the request packet and into appropriate variables in its address space (step 6). This action, the inverse of marshalling, is called *unmarshalling*. The server can then actually execute the request, performing whatever actions are required with the data provided (step 7). After executing the request, the server then must build a *response packet* (step 8), by marshalling the results data (o_1, o_2, ... o_n) from variables in its address space to a contiguous buffer. Finally the server sends the response packet back to the client over the connection (step 9). The client, which has been patiently waiting for the response, at last receives it over the connection (step 10). The client's last step (step 11) is to unmarshall the results data from the response packet to appropriate variables in its address space.

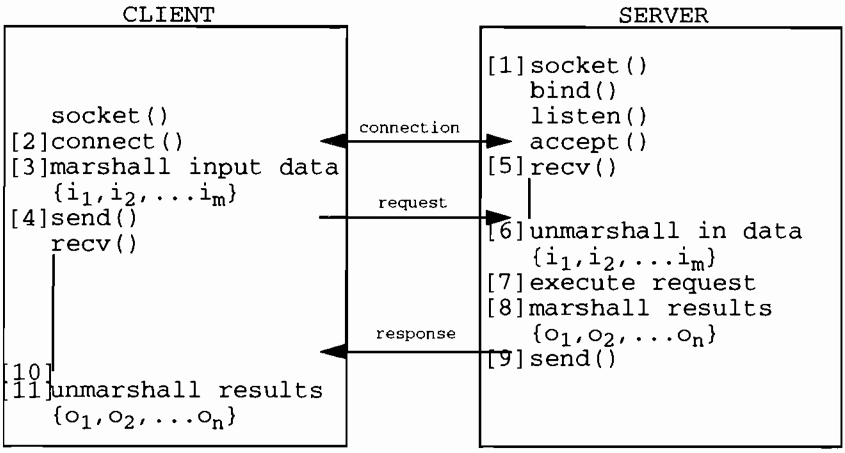


Figure 1. A Client-Server Exchange using BSD Sockets

As can be seen from the preceding discussion, the IPC method requires the application programmer to manage the network interface between the client and server: creating and connecting sockets, sending and receiving packets, and recovering from errors. The application programmer must also build and interpret the request and response packets for each exchange between the client and server. This can add appreciable overhead to the coding of a client-server application. It may be desirable to automate the generation of this overhead code and to hide the details from the application programmer. This can be done using the RPC technique.

REMOTE PROCEDURE CALLS

A service request between a client and a server can readily be formulated as a procedure call. Using the terms from the IPC example in Figure 1, the procedure call would be:

```
        service_request(i1,i2,...im,o1,o2,...on);
```

where $i_1, i_2, \ldots i_m$ are input parameters, and
$o_1, o_2, \ldots o_n$ are output parameters.

Using the RPC technique, the application programmer can simply code a procedure call for the service request, and the RPC software will automatically take care of the network interface and the marshalling and unmarshalling of the input and output parameters. This can greatly simplify development of a client-server application.

There are several current and future software products that provide RPC mechanisms. This paper discusses the Distributed Computing Environment (DCE)[7,8] from the Open Software Foundation (OSF). The Open Software Foundation acquires, through its Request for Technology process, key software components for operating systems and application development and integrates these components into vendor-neutral packages. Vendors then port the packages to their own platforms. Table 2 shows a number of the vendors that have announced plans to port DCE to their platforms. A wide variety of DCE implementations will assure multivendor interoperability for DCE-based client-server applications.

Table 2: OSF DCE Ports

VENDOR	PLATFORM	OPERATING SYSTEMS
HP	HP3000 S900 HP9000 S700/800	MPE/iX HP-UX
DEC	DECstation VAX	Ultrix, OSF/1 VMS
IBM	RS/6000 Mainframes	AIX, OSF/1 MVS (SAA)
Transarc	SPARC	Solaris
Gradient	Intel 80x86 PCs	Windows 3.x

The Network Computing System (NCS)[9] was developed by HP Apollo and has been ported to a number of platforms including the HP9000 S300/400/800 and the HP3000 S900. NCS provides an RPC mechanism very similar to DCE RPC; in fact, DCE RPC is NCS Version 2.0 (which was a collaborative effort between HP Apollo and DEC).

An RPC mechanism is also a part of SunSoft's Open Network Computing (ONC) [10, 11]. This RPC is used by the Network File System (NFS), which has been ported to many platforms including the HP 9000 and the HP 3000. The DCE RPC and ONC RPC are not currently compatible with each other.

Figure 2 shows a client-server exchange using DCE RPC that corresponds to the IPC example in Figure 1. The calls coded by the application programmer are in elite, while the calls performed by RPC are in italics. The server starts by declaring which protocol it will use (step 1), which creates a socket to receive connection requests. The server then declares which remote procedures it can execute (step 2: *interface registration*) and waits for incoming remote calls (step 3). The client indicates the server to be used (step 4: *binding*) and calls the remote procedure (step 5). The establishment of the connection between the client and the server, the marshalling and unmarshalling of the input and output parameters, and the sending and receiving of the request and response packets, are all done automatically by RPC. RPC in the server calls the application-supplied **service_request()** routine (step 6) to execute the request.

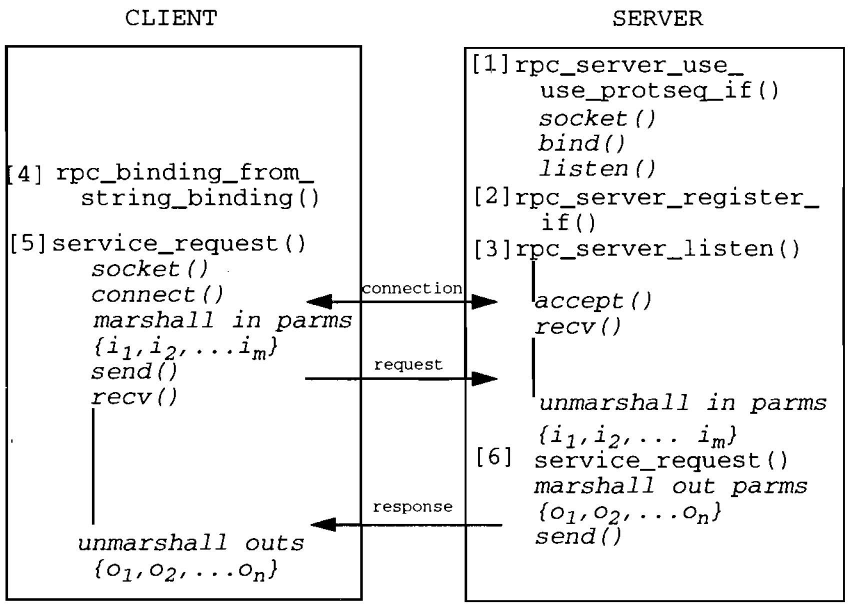


Figure 2. A Client-Server Exchange using DCE RPC

INTERFACE DEFINITION LANGUAGE

To automatically generate the code to handle remote procedure calls, RPC must know certain things about the procedures and their servers. For example, it must know the types and lengths of each parameter value it has to marshall or unmarshall. The application programmer provides this information using the Interface Definition Language (IDL). Each IDL file describes a related collection of remote procedures, called an *interface,* that a server can execute on behalf of clients. First the IDL file provides some general information about the interface and its server(s), such as an identification of the interface (called the *interface UUID*), the version of the interface, and (optionally) a well-known address to be used by the server. Then a definition is given for each procedure in the interface, including the procedure name and the types and usage of each parameter. The procedure definitions are based on the function prototypes of the C language, with extensions for information required by RPC. The extensions are enclosed in square brackets and include things like whether the parameter is used for input or output (or both) and where the actual length of an open array is to be found.

```
[uuid(f9f6be80-2ba7-11c9-89fd-08002b13d56d),
version(1.0),
endpoint("ncacn_ip_tcp:[6680]")]

interface array
{
void sort([in] handle_t ah,
          [in] long array_size,
          [in, out, size_is(array_size)]
               float array[]);
}
```

Figure 3. A Simple Interface Definition in IDL.

A simple example of an IDL file is shown in Figure 3. This defines an interface named **array**, containing procedures for array manipulation which might be executed on a fast computation server. The interface is assigned an unique identifier (the **uuid**, which can be generated by the **uuidgen** program) and a version number. The **endpoint** specifies the TCP port number to be used when connecting to the server.

The interface shows one procedure, **sort()**, for sorting an array of floating point numbers. **sort()** has three parameters. The input parameter **ah** is a handle that identifies the server to which the client has bound, through a previous **rpc_binding...()** call. The input parameter **array_size** specifies the actual number of values in the array to be sorted. The last parameter **array** is the array of values to be sorted. It is used for both input (the unsorted values) and output (the sorted values). The procedure has no (**void**) return value.

IDL COMPILER AND CLIENT AND SERVER STUBS

Once an interface has been defined by an IDL file, the client and server code to handle the remote procedure calls in the interface can be generated automatically. The tool used for this is the *IDL Compiler*, which takes as its input the IDL interface definition and produces three files of C language source statements: the *client stub source*, the *server stub source*, and the *interface header file*. For each interface procedure, the client stub contains code to marshall input parameters, send request packets, receive response packets, and unmarshall output parameters. Similarly, for each interface procedure, the server stub has code to receive request packets, unmarshall input parameters, call the actual procedure, marshall output parameters, and send response packets. The header file provides information about the interface that is used in the client and server stubs. After they are produced by the IDL compiler, the client and server stub source files must be compiled with the C compiler to produce objects for use in the client and server programs. (The IDL Compiler can do this automatically.) This is illustrated in Figure 4. The modules generated by RPC are indicated by italics; the other modules are supplied by the application programmer.

In addition to the IDL file defining the interface, the application programmer provides three other modules: the *client application source*, the *server main source*, and the *interface manager source*. The client application source finds and binds to the server and makes the remote procedure calls. The server main source performs several tasks to set up the server, such as specification of supported protocols and registration of the interface. The interface manager source provides the actual procedures to be executed by the server: the real work requested by the client. These sources can be written in any programming language that provides a procedure calling mechanism, for example, C, Pascal, or Cobol. (But if they are written in a language other than C, the application programmer may have to provide some additional C code to handle binding between the client and server, which uses the interface header file.) After compilation by the appropriate compiler(s), the client application object is linked with the client stub object and the RPC Runtime library to produce the client program, and the server main and interface manager objects are linked with the server stub object and the RPC Runtime to make the server program. (The RPC Runtime Library contains the bulk of the code to manage the network interface, to send and receive packets, and to keep track of the state of the binding between the client and the server.)

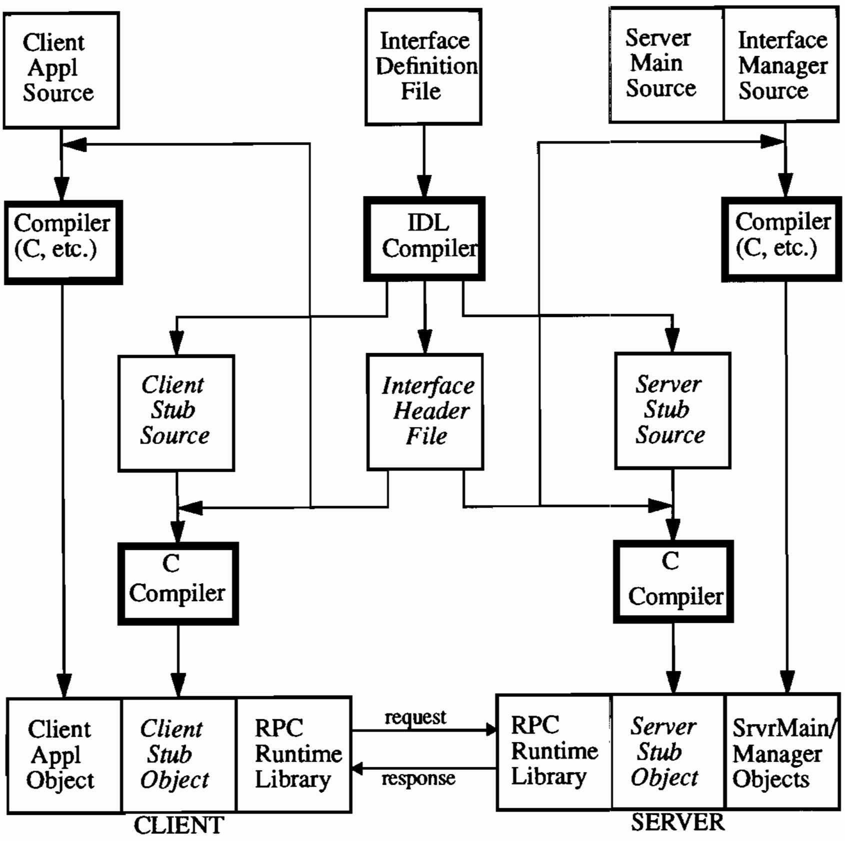


Figure 4. Generation of Client and Server Stubs Using the IDL Compiler

COMPARISON OF IPC AND RPC FOR AN APPLICATION

To compare the effort required by the IPC and RPC methods, a sample client-server application was coded using each technique. The application provides a simple print server. The **netp** client is run with a list of files to print (e.g. **netp file1, file2, ... filen**). It reads some configuration data (the server location and print device parameters) from the file **netp_conf**. The client connects or binds to the **netp_server** and sends a **netp_open** request to the server to open a print device with the given parameters. Then, for each file to be printed, the client sends a **netp_file** request, containing the file name, size, and contents. The server receives each request and writes the file contents, with a header and pagination, to the print device. For the last file to be printed, the client sets a flag in the **netp_file** request to tell the server to close the print device. Figure 5 illustrates the operation of the client and server.

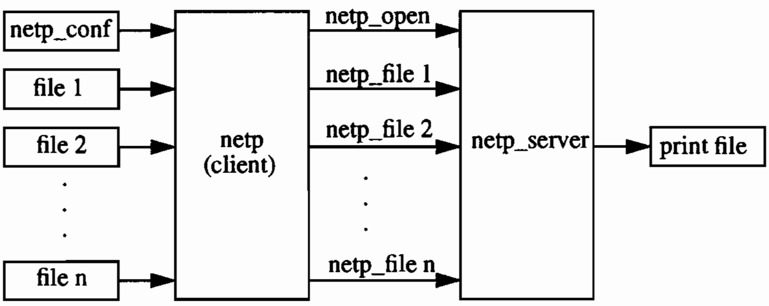


Figure 5. Netp client and server

Two versions of the **netp** and **netp_server** programs were coded in C, the first using BSD Sockets, and the second using DCE RPC. Tables 3 and 4 show a comparison of the number of non-comment lines of code (NCLOC) needed for the various client and server tasks in the IPC and RPC implementations. In the client, the IPC socket setup is comparable to the RPC calls to bind to the server, and the IPC marshalling/unmarshalling and send/receive calls are comparable to the remote service calls for RPC. In the server, the IPC socket setup is comparable to the RPC server main initialization (declare protocol, interface registration, and listen), while there is no RPC code comparable to the IPC marshalling and send/receive calls. Application logic is the code to perform the print tasks: reading of the files in the client and writing to the print device in the server. For the RPC implementation, the server application logic becomes the interface manager module (that is, the implementation of the remote procedures).

Table 3: Comparison of IPC and RPC Versions of the Netp Client

IPC	#ncloc	RPC	#ncloc
socket setup	38	bind to server	17
marshalling & send/recv	102	remote calls	14
application logic	104	application logic	106
TOTAL	244	TOTAL	137

Table 4: Comparison of IPC and RPC Versions of the Netp Server

IPC	#ncloc	RPC	#ncloc
socket setup	49	server setup	18
marshalling & send/recv	104		
application logic	133	interface manager	143
TOTAL	286	TOTAL	161
		IDL File	19

One aspect of BSD Sockets (and also NS NetIPC) that the IPC client and server must deal with is that the TCP connection is in stream mode. This means that the **send**() and **recv**() calls produce and consume a continuous stream of bytes across the connection. There are no message boundaries. One **send**() call is not guaranteed to transmit an entire packet, while a **recv**() call may not receive an entire packet. Factors like maximum network packet size and flow control may effect the calls. Consequently, two procedures **send_pkt**() and **recv_pkt**() were added to the IPC client and server to ensure that entire packets are transmitted. This problem is solved internally by the RPC Runtime Library.

The requirement that the server handle multiple clients concurrently has implications for both the IPC and RPC implementations. In both cases the server must maintain separate context, including the print file descriptors and page layout parameters, for each active client. For the IPC version of the server, this problem was solved neatly (at least for Unix or POSIX-compliant systems) by having the server fork a copy of itself for each client after it accepts the connection. The new server copy inherits the connection and is then dedicated to that client. (This could be done on an MPE/iX system with NS NetIPC by having the server create a copy of itself and call **ipcgive**() on the connection, and have the new server copy call **ipcget**() for the connection.) The problem was solved for the RPC implementation using the *context handle* concept. The **netp_open**() procedure allocates a record to hold the context information for the new client and passes the pointer to the record back to the client as a context handle. The client includes the context handle in subsequent calls to **netp_file**() so the server can locate the client's context record. The server also supplies a *context rundown routine* that RPC will call if the connection to the client is lost. This routine cleans up the client's context, including closing the print file.

ADDITIONAL BENEFITS OF RPC

In addition to the automatic generation of client and server stubs to handle the marshalling/unmarshalling of data and the interface to the network, RPC provides a number of features that make developing complex distributed applications easier. These advanced features are largely absent in IPC products.

Protocol Choices

There may be more than one choice for the underlying transport protocol used in a client-server application. For instance, the application might use the TCP connection protocol if there were many long requests from a client to a particular server, or the UDP datagram protocol if there are a lot of short requests from a client to many different servers. RPC allows the server to specify which protocols it is able to use, and the client to choose the best protocol for the application (or use a default). Since the UDP protocol is unreliable, RPC includes a datagram protocol to detect and recover from lost, mis-ordered, or duplicated packets. With IPC, the application may be have be modified to switch from one protocol to another. If UDP is used, the application must provide its own recovery mechanisms.

Server Location Methods

In the preceding discussion, the important step of "finding the appropriate server" was largely ignored. RPC provides three ways that a client can locate a server:

Well Known Address: This is the simplest technique, where the server uses a fixed endpoint and the client knows the node location of the server and the endpoint. This method is not very flexible, requiring that the endpoint be uniquely assigned to the server. IPC mechanisms usually provide this method.

Endpoint Mapping: With this method, the server uses a dynamically assigned endpoint, which it registers with a special server on its node, called the *Endpoint Mapper*. The client knows only the node location of the server. With the RPC datagram protocol, the first remote call is sent to the Endpoint Mapper, which forwards it to the endpoint of the target server. The response to the first call contains the server's endpoint, so the client can send subsequent calls directly to the server. With the RPC connection protocol, the client's RPC Runtime retrieves the target server's endpoint from the Endpoint Mapper (through a remote call using the RPC datagram protocol) before establishing the connection. Some IPC mechanisms provide this feature (e.g., the Socket Registry facility of NS/3000 NetIPC and Lan Manager Named Pipes); others do not.

Name Service Interface: It is preferable that the client not have to know the node location of a server at all. Instead, the client should be able to find the server based on an application-defined name. Then the server can be moved to a new location without the client's knowledge.The RPC Name Service Interface (NSI) provides this facility. The server registers its name and location through the NSI, and the client uses the name to bind to the server. (This can either be coded explicitly using the **rpc_ns_binding...** calls or it can be done through *automatic binding*.) The NSI uses the DCE Cell and Global Directory Services (described later). In addition to name space entries for servers, interfaces, and objects, the NSI name space provides entries for *groups* of related servers and for *profiles* of servers based on user requirements.

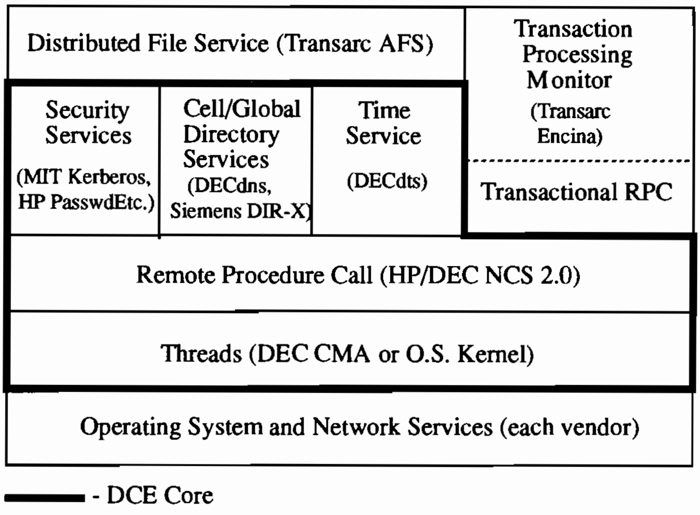
Data Conversion
As was noted previously, many client-server applications will run on different vendor systems, with different architectures and data representations. One example is the ordering of bytes in integer data (the so-called *big-endian vs. little-endian* problem.) Some machines, like the HP 9000 and the HP 3000, represent integers with the most significant byte first. This order is reversed, with the least significant byte first, on other systems such as the VAX and the Intel 80x86. Data to be exchanged between such systems requires conversion at some point. RPC uses the concept of "receiver makes it right", where the receiver of the data converts from the sender's representation to its own. This is done automatically by the client and server stubs, using the Network Data Representation (NDR) protocol. With IPC, this conversion must be done explicitly by the application programmer.

Object-RPC Mapping
It can be very useful to define an interface that can operate on a number of related types of objects. As examples, a print server might work with various types of printers, and a file server might provide uniform access to different kinds of files. Things of interest in a client-server application can be defined as *objects* and assigned identifiers called *object UUIDs*. Objects can be grouped into *types*, which are assigned *type UUIDs*. RPC allows the interface procedures to be mapped to different *manager* procedures based on a particular type of an object. The association of an object with a type (based on their UUIDs) is done during the initialization of the server, as well as the mapping of the interface procedures to the managers for the type. When the client binds to the server, it can specify the UUID of the object it intends to use.

Suppose, in the file server example, the interface defines general procedures for opening, reading, writing, and closing a file. Types of files may be flat files or relational databases. A server registers each file it can access, along with the type of the file. A client binds to the server, requesting access for a particular file. When the client executes a file access call, the server maps it into the particular routine for the type of file, say, **netfile_open**() is mapped into **flatfile_open**() or **database_open**(). In this way, the client can use the interface to access files in a general way without being aware of the underlying file type or implementation.

OTHER DCE COMPONENTS

While the RPC mechanism described here provides considerable benefit in itself, it will be just one component in an overall system for distributing processing offered by OSF DCE and related products. Figure 6 (a variant of the standard DCE picture) shows how these components fit together, along with the suppliers of the original technology for each component. There is a subset of the components, the DCE Core, that is needed for general distributed processing. All vendors providing OSF DCE will provide the core components. Most of the components use RPC for client-server communication. In turn, RPC uses the other core components for various tasks, as discussed below.



Distributed File Service (Transarc AFS)
Transaction Processing Monitor (Transarc Encina)
Security Services (MIT Kerberos, HP PasswdEtc.)
Cell/Global Directory Services (DECdns, Siemens DIR-X)
Time Service (DECdts)
Transactional RPC
Remote Procedure Call (HP/DEC NCS 2.0)
Threads (DEC CMA or O.S. Kernel)
Operating System and Network Services (each vendor)
- DCE Core

Figure 6. OSF DCE and Related Components

Threads

A server may be required to concurrently service a number of client requests. If a request takes a long time or causes the server to block, it will delay the other requests. The solution to this problem is the concept of *threads*. Each thread is an independent path of execution within the server. Each thread has its own local variables (execution stack) but shares global variables, open files and sockets with the other threads in the process. A server may have a number of threads to handle concurrent requests. DCE supplies a user-space threads package based on the DEC Concert Multithread Architecture (CMA) with a POSIX pthread interface[12]. Some operating systems (OSF/1, MPE/iX) will use their own kernel-space threads in place of CMA. RPC uses threads to concurrently manage network connections, execute calls, and handle timing events.

Cell and Global Directory Services
DCE provides two directories to map names to objects. The Cell Directory Service (CDS), based on the DEC Distributed Name Service (DECdns), handles naming within an administrative group of systems called a *cell.* A cell may range from a local work group of several workstations to a campus of hundreds of machines. CDS provides high availability by replicating its directory database on different systems and high performance by locally caching frequently used name entries. The Global Directory Service (GDS), based on Siemens DIR-X implementation of the CCITT X.500 directory standard, provides a world-wide name space. RPC uses CDS and GDS (through its Name Service Interface) to hold entries mapping interfaces and objects to the appropriate servers.

Security Service
Security is an important but difficult requirement for distributed applications. DCE incorporates a number of security features, including authentication, authorization, and encryption based on the MIT Kerberos project[13] and the HP PasswdEtc product. *Authentication* allows *primaries* (say, a user or a non-interactive server) to reliably establish their identities (by, for example, logging on with a password known only by a user and the Authentication Service). *Authorization* allows a server to control access to objects based on a client's primary identity. DCE Security uses *Access Control Lists (ACLs)* which define the privileges each primary possesses, and *Privilege Attribute Certifications (PACs)* to reliably communication the primary's identity and group memberships. RPC uses these features to provide several levels of security for the communications between the client and server. These range from no security, to authentication between the client and server, to encryption of each packet between the client and server. The application developer can then chose the level of security (and the associated overhead) required by the application. (Note that since the encryption uses the Data Encryption Standard (DES), it is not available outside of the United States.)

Time Service
The clocks on different systems in a network may differ from one another, possibly leading to problems for distributed applications. The DCE Time Service, which is based on DEC's Distributed Time Synchronization Service (DECdts), ensures that the time values used by clients and servers are accurate and consistent. It coordinates the clocks between systems and rejects those clock times that are out of line with the general consensus. Applications may get the correct time through the UTC (Universal Time Coordinated) interface.

Distributed File System
Easy access to files on different systems in a network is important to users and

applications. DCE provides a Distributed File System that allows files to be accessed by normal POSIX-style file calls (**open**(), **read**(), **write**(), etc.) anywhere from the network. DFS allows multiple readers and writers of a file, controlling access through a token scheme. DFS uses CDS and GDS to provide network-wide, location- independent file names. (Note that DFS is not a DCE Core Service and so may not be available on all DCE-compliant systems.)

Transaction Support
Integrity of transactions is very important for on-line transaction processing (OLTP) applications. This means that if a transaction is committed, all actions of the transaction, including remote procedure calls, take effect. If a transaction is aborted or rolled back, all actions, including remote calls, must be undone. This is a particularly difficult problem for distributed applications. The Encina product from Transarc Corporation provides a transaction monitor built on top of DCE, with transactional extensions to RPC. Encina has been endorsed by a number of important vendors, including IBM and HP, which plans to provide Encina on both the MPE/iX and HP-UX platforms. (Note that Encina is not part of the DCE offering and so may not be available on all DCE-compliant systems.)

Comparison of RPC Product Features
Table 5 shows features of DCE, NCS (Version 1.5.1), and ONC that are approximately equivalent. In some cases (for instance, the DCE Name Service Interface and the ONC Network Information Service), the equivalence is very approximate.

Table 5: Comparison of RPC Products

DCE	NCS	ONC
IDL Compiler	NIDL Compiler	Netwise RPC Tool
Endpoint Mapper	Local Location Broker	Port Mapper
Name Service Interface	Global Location Broker	Network Info Service
Network Data Rep NDR	Network Data Rep NDR	External Data Rep XDR
Objects, type managers	Objects, type managers	N/A
POSIX Threads	(Domain NCS only)	N/A
Kerberos Authentication	N/A	Secure NFS
Time Service	N/A	Time Daemon
Distributed File Service	N/A	Network File Service

CONCLUSIONS

RPC provides a powerful paradigm for developing client-server applications. Using the Interface Definition Language, the IDL Compiler, and the RPC Runtime Library, an application developer can produce distributed applications with much less effort than by using the lower-level IPC paradigm. RPC provides many useful automatic features, including choices for server location, protocol selection, and data conversion. Furthermore, RPC is integrated with the OSF Distributed Computing Environment, which will be supported by many vendors. RPC and DCE ease the task of developing applications that interoperate between different operating system and architecture platforms.

REFERENCES

1. Stevens, R. W., **Unix Network Programming**, Prentice-Hall, 1990.

2. Leffler, S. J., McKusick, M. K., Karels, M. J., Quarterman, J. S., **The Design and Implementation of the 4.3 BSD Unix Operating System**, Addison-Wesley, 1989.

3. Faulkner, K., Knouse, C., Lynn, B., *Network Services and Transport for the HP3000 Computer*, **Hewlett-Packard Journal**, Vol. 37, No. 10, October, 1986.

4. **NetIPC 3000/XL Programmer's Reference Manual**, Hewlett-Packard Co., Part No 5958-8600.

5. **Interprocess Communication Programmer's Guide**, Hewlett-Packard Co., Part No. 32650-90019.

6. Ryan, R., **Microsoft Lan Manager: A Programmers Guide, Version 2**, Microsoft Press, 1990.

7. **OSF Distributed Computing Environment Rationale**, Open Software Foundation, 1990.

8. **Distributed Computing Environment Application Development Guide**, Open Software Foundation, 1991.

9. Kong, M., Dineen, T. H., Leach, P. J., Martin, E. A., Mishkin, N. W., Pato, J. N., Wyant, G. L., **Network Computing System Reference Manual**, Prentice-Hall, 1990.

10. **ONC/NFS: A Technology Guide to Distributed Computing**, Sun Microsystems, Inc., 1991.

11. Santifaller, M., **TCP/IP and NFS: Internetworking in a Unix Environment**, Addison-Wesley, 1991.

12. IEEE Standard 1003.4a, **Threads Extension for Portable Operating Systems**, The Institute of Electrical and Electronics Engineers, Inc., 1990.

13. Kohl, J., Neumann, B. C., **The Kerberos Network Authentication Service**, MIT Project Athena, Network Working Group, 1990.

Caveat Emptor:

What Buyers and Developers of Client/Server Applications Need to Know

INTEREX '92 Paper # 3851

by

Michael A. Bailey

Hewlett-Packard Company
501 56th Street
Charleston, West Virginia 25304
(304) 925-0492
mbailey@a2614uxa.msr.hp.com

Abstract

To be successful in implementing or developing an application using client/server technology, the IS professional must understand the new parameters of the client/server model. Designing client/server applications is a rather new skill; developers and implementors must be concerned with issues that never arose in the days of Terminal/Host applications, including network traffic minimization, data security and integrity assurance, networked system performance bottlenecks and data and processing distribution. In addition, industry standards such as the Open Software Foundation's Distributed Computing Environment (DCE) must be considered when making client/server design or purchasing decisions. Correct designs lead to applications that are flexible, easy-to-use, make good use of available computing resources and can grow with the customer's business. The paper takes a comprehensive look at issues that confront designers and purchasers of client/server solutions.

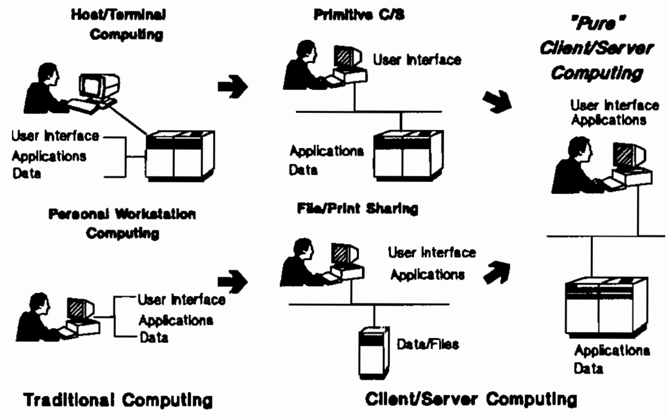
Introduction

Client/server computing has received much attention over the last two years and has been called *the* computing architecture of the 1990s. Nearly every software supplier purports to have a solution that is based on the client/server model and most new in-house development projects are attempting to use the technology. All "client/server" applications are not created equal, however. To be successful in implementing or developing an application using this technology, the professional must understand the new parameters of the client/server model so that they may discern well-designed client/server applications from those whose design's limit their effectiveness and make them difficult to manage. A number of technical issues will be explored in the following discussion.

Ancient History and the PC Revolution

For a moment, however, ignore the technical aspects of the client/server model. To create or implement useful applications using this new paradigm, it is necessary to understand the environment from which it arose. Until approximately 1981, computer users interacted with a terminal attached to a large, expensive mainframe or minicomputer. While users had access to information via the terminal, it was in a format designated by MIS and could be massaged only by applications that had been requested of MIS. Clearly users were operating within the domain of MIS. The advent of the PC revolutionized computing forever. The standalone PC became a choice for many users. Applications were typically more friendly than their terminal-based counterparts and tools like Visicalc® and Lotus 1-2-3® allowed users to view and use information on *their* terms. The problem, of course, was the PC became an island of information. Until about 1985, terminal/host systems and standalone PCs were the only options generally available, and each suffered from their own unique problems, as well as one in common. In both cases, the application, data and user interface all resided on a single system (Fig. 1).

Figure 1



Primitive client/server applications began to appear with the advent of the Local Area Network. PCs could be attached to the LAN and perform terminal emulation to the mainframe or minicomputer. It was even possible to run part of the user interface on the PC, as HP's VPLUS for Windows product does. On the other hand, the standalone PC could be attached to the network and use a server for disk and printer sharing. Some crude multiuser applications were developed that allowed multiple clients to access a common database on the server with record locking enforced. These primitive client/server mechanisms were improvements over their predecessors, but each still had problems. Placing the user interface on the client was of some help in integrating the mainframe or minicomputer application with PC-based applications, but was of little use in integrating those that ran only on the host. In the file server environment, the server provided no intelligence; it simply processed file requests. Thus, every file access was performed across the network, elongating response times and generating extremely high network traffic with a rather modest number of users. Nor did file servers resolve the "island of information" problem; they simply created a larger island. Clearly neither of these environments adequately met user's needs.

Evolution of the Revolution

Enter client/server computing, the evolution of the PC revolution. Eleven years after the first PC was shipped, millions of workers have had no other exposure to information systems. They view their PC as the platform to generate and access the information needed to perform their jobs. They want to create and access information using a common toolset, and they care not the slightest about databases, networks, or three generations of COBOL code that comprise today's legacy systems. This brings us to an extremely important point: *Client/server computing is a demand of the user community, not the idea of MIS.* Users may not know what to call it, but they will surely recognize its presence or note its absence. It is we, the Information Systems professionals, that must respond to the new requirements of the user community. Any developer or purchaser of a client/server system that does not recognize this fact will fail.

Justified by Cost?

One further point regarding the client/server movement should be noted before we move to the technical issues. Often it is incorrectly assumed that client/server applications should be developed simply because they reduce costs by taking advantage of inexpensive desktop computing power. This is an assumption that will lead, again, to failure. Client/server computing requires changes in the way a company does business; workers have greater latitude to make decisions because they can access and use information *on their own terms using their own tools.*

Computing is becoming less expensive regardless of the chosen methodology. Client/server computing may have still lower costs, depending on the degree of change it requires in the organization, but the true benefit to the model is the empowerment it brings to users. Employees empowered with information create a productive, competitive staff. Organizations unwilling to make the environmental and behavioral changes the model expects would be wise to maintain a traditional approach to information processing.

Having heard the sermon, let us now concentrate on some of the technical issues that surround client/server computing. Several questions may come to the mind of someone new to the client/server model. Among them:

I run Rbase from a server - don't I already have a client/server application?
How does one develop a client/server application?
Is one kind of database any better than another?
Won't a client/server app flood my network moving data around?
There are so many network protocols, do I need to pick one? Which one?
Will OSF's Distributed Computing Environment help me?

The sections ahead should answer all of these questions and more.

Client/Server Definition

The term "client/server" surely vies for the title of most over-used term in the industry. Nearly every supplier of software touts their "client/server" implementation, with "client/server" being defined by the vendor. Thus, one of the primary issues inhibiting the growth of client/server solutions is the fact that many purchasers are not sure what it is[1]. At the risk of adding to the confusion, consider one more definition: client/server computing is simply the interaction between intelligent processes, whereby a *client* process makes requests and a *server* process answers those requests. Nowhere in the definition are multiple platforms mentioned, although the implication is strong. The key point is that each process is intelligent. This is why a PC database in conjunction with a file server does not fit the model. All of the application code runs on the local PC and the data files are simply accessed across the network. A request by the PC for records with a particular field value requires the server to move every record in the file across the network *because the intelligence to examine field values resides in the PC, not the server.* An intelligent server process would perform the lookup and return only those records that match the selection criteria.

[1] Strehlo, Kevin, "Client-Server Deep Throat", *DBMS*, June, 1991, pp. 8-10

Development Methodologies

A reasonable observation at this point would be that a client/server application must be considerably more difficult to create than the vintage HP3000 application, which consists of a COBOL program that provides a user interface and data access via calls to VPLUS and TurboIMAGE intrinsics. Client/server development is certainly different, but not necessarily more difficult. The new paradigm enforces a natural discipline not found in traditional systems. The idea of breaking up the user interface, data access and application logic forces the developer to split the application into smaller, more cohesive and less interdependent (low level of coupling) modules than might otherwise be the case. In software engineering terms, modules with high cohesion and low inter-module coupling are easier to develop and maintain.

It does not stand to reason that conventional development methodologies will suffice in the client/server environment. Traditional systems exist in two dimensions: it must be determined how broad the application will be, with respect to the target problem (ie. general ledger or a complete accounting package), and to what depth the application will be written (ie. include general accounting principles or the nuances of cost and managerial accounting). Client/server applications add a third dimension: location or, in question form, where will the processing take place? The development methodology chosen must account for this third dimension.

The type of processing the application is to perform will also influence the choice of development methodology. Applications which are more operational in nature (that is, provide transaction processing functions) require a different development technique than those created primarily for decision support.[2] Inmon's description of a development methodology for an operational system bears some resemblance to conventional methods, in that he speaks in terms of Entity-Relationship Diagrams, Data Flow Diagrams, pseudocode and walk-throughs. It is a paper- or CASE tool-based system appropriate for more complex client/server systems. His decision support methodology, as one would expect, focuses more on the *data* issues of the problem than on the processing specifications. Like its counterpart, it is appropriate for large applications.

An alternative approach, focusing on rapid application development, is presented by Louderback.[3] CSIRD, an acronym for "Client/Server Iterative Rapid Development", is based on a small team of information processing specialists and

[2] Inmon, W. H., *Developing Client/Server Applications in an Architected Environment*, (QED Technical Publishing Group, 1991), p. 104.

[3] Louderback, James, "The CSIRD Methodology", *DBMS*, June, 1991, pp. 46-52

end-users building a complete system by designing, developing and implementing different modules in parallel. It is intended to create a fully-functional application in weeks rather than months. Appropriate for small to medium-sized systems, it is best suited for decision support applications and those that do not require complex processing.

Developers of client/server systems view development techniques from a much different perspective than do buyers. Software suppliers or in-house developers must be confident they can make a given approach work in their environment. A buyer of a client/server system can make a more informed decision by investigating the development techniques used to create the candidate applications. If either a developer or buyer lacks experience in client/server development or acquisition, they would be wise to seek assistance from a knowledgeable consultant. Changing the way information is processed in a organization will, no doubt, be a highly visible undertaking. There may be only one chance for success.

Implementation Issues

In addition to the aforementioned development methodologies, the informed developer or buyer will consider a number of other issues when analyzing implementation options. Some of these will be mentioned momentarily, but first a point about *how* to make a wise choice. Any good decision will, among other factors, be based on the anticipated scope of the implementation. The industry is notorious for underestimating the size, longevity and inherent limitations of a system under development (the phenomenally low sales estimates of the original IBM PC and the infamous 640KB addressing limitation of the same system come to mind). As Eero Saarinen said, "Always design a thing by considering it in its next larger context - a chair in a room, a room in a house, a house in an environment, an environment in a city plan". Ask the "What if ..." questions: What if the anticipated number of transactions doubles? What if another graphical interface steals the users' hearts? What if this system is in place and entrenched in 10 years? What if Corporate chooses another network transport service? One does not have to have definitive answers to the questions, only to have factored them into the decision process.

Many of the issues confronting the buyer or developer can be lumped into the following categories: hardware and software platforms for the server and client, database platforms, networking infrastructure and personnel requirements. These issues are intertwined in the real world, but they will be discussed separately here so that the substance of each issue is clear.

Platform Options: Server

In the age of commodity hardware, platform choices should be simple, right? Nice thought. There are so many options, each offering different features, that making a decision can be daunting for the uninitiated. Not all software environments are available for each hardware platform, therefore a choice of either type reduces the options for the other platform. There are, generally, three hardware platforms for servers: PCs (Intel platforms), workstations (UNIX-based) and minicomputers.

PCs are the traditional server, running dedicated Network Operating Systems like Novell NetWare or non-dedicated Network Operating Systems like LAN Manager running on OS/2. These systems offer a rich set of client/server applications, as well as good file and peripheral sharing. They are handicapped by a lack of expandibility and, even with the 486 chip, comparatively low performance. Additional expansion and performance can be achieved by adding servers to the network, but this can add a much higher level of complexity if the application cannot be easily split. Dividing one logical database among servers creates a distributed database environment, and requires the integrity, fragmentation, allocation and optimization issues of this environment be addressed. The PC platform is a good choice for many smaller departmental applications.

UNIX workstations are a relatively new and fast growing segment of the server market. The openness of the operating system has enabled a large number of vendors to supply industrial-strength database systems and server software. These include Oracle, Ingres, Informix and Sybase. The high raw performance and price/performance ratio of Hewlett-Packard's Series 700 workstations, combined with the aforementioned database systems or HP's ALLBASE/SQL, makes for an excellent server. The HP workstation platform also holds promise for users that have committed to Novell NetWare. The recent cooperation between HP and Novell to develop a native version of NetWare for the Series 700 means NetWare users can look forward to taking advantage of HP's industry-leading RISC architecture.

The workstation platform is, by definition, somewhat limited in I/O expandibility. With the advent of disk array technology, workstations can have enough disk capacity for the majority of applications. However, workstations simply do not have enough "real estate" to handle a large number of peripherals or network interfaces. The workstation is an excellent server choice for applications that require robust database and system administration functionality and high performance, but do not require a high degree of expandibility in a server.

Minicomputers are expanding their role as the definitive departmental computing platform to include server functionality. The extremely high degree of scalability (as evidenced by HP's HP3000 and HP9000 Corporate Systems, which support from a handful to thousands of users) offered by this platform makes the minicomputer a good choice as a server. The minicomputer offers a high level of connectivity and expandibility. With support for Portable NetWare and LAN Manager, as well as industry-leading database management systems, the minicomputer is positioned to allow its customers to *evolve* into client/server computing. Only a minicomputer can act as database server, file/peripheral server, gateway to mainframe data, and On-Line Transaction Processing engine to a large number of users. It should be noted, however, that the multipurpose nature of the minicomputer means it may not always offer the highest possible performance for a particular application as one of the other server types.

Platform Options: Client

Because the user interface in the client/server environment resides on the client, the choice of a client platform is often based on personal preference. The contenders include PCs, Apple Macintosh, and UNIX workstations. Each of these have their ardent supporters, even while the differences between them are becoming less dramatic. The workstation certainly is the high performance platform, the PC has the unequalled availability of off-the-shelf software and the Macintosh has a perceived ease-of-use advantage.

For the developer or buyer of a client/server system, it is imperative that the end user's preference for a client be thoughtfully considered. An outstanding client/server system, created specifically for UNIX clients, will never be considered by users with a strong allegiance to the Macintosh. If multiple client platform requirements are even remotely possible, the wise developer will choose a cross-platform development tool to create the system. Multiple platform support should be near the top of the buyer's list of questions for a supplier.

It is now commonly accepted that the client/server environment demands a graphical user interface on the client platform. The object, again, of client/server computing is to empower users with information such that they become more productive and increase the quality of the services they provide. GUIs are considered essential to this end. Another quality of the client which is highly desirable is some level of multitasking. Many workers must have the capability to switch between several applications. Indeed, a graphical interface and multitasking can help a user integrate disparate applications *at the desktop*. These two features of the client allow client/server computing to bring the promise of information technology to every user.

The battle between the Macintosh, UNIX, DOS and OS/2 has been fought on the pages of nearly every industry publication for several years. This paper will not attempt to provide the last word. The point to be made here is this: consider the strengths and weaknesses of the platforms with respect to both current and anticipated user needs, and choose client development tools that support multiple platforms if such a requirement exists.

Database Platforms

There is no more important facet in determining a client/server environment than the data management platform. Some may argue a database management system is not necessary; that a client/server system can consist of a series of systems connected together via a distributed file system such as the Network File System (NFS). The problem with this approach is that it makes the server a comparatively unintelligent member on the network. As we will see, intelligence on the server provides higher levels of performance, functionality and integrity. Make no mistake; a database management system is required to implement true client/server computing.

As with hardware and software platforms, the choice of a database management system is complicated by the wide range of alternatives. Every server platform offers a large number of DBMS solutions, but every solution is not available on all servers. The matter is even more complicated in a heterogeneous environment, which brings with it the issue of interoperability among DBMS platforms. Until very recently, it has been up to the individual DBMS vendors to provide interoperability amongst themselves. This has led to an interoperability "minefield" through which purchasers must carefully navigate. Only a well-informed buyer can choose a DBMS platform that will meet his needs over the long term.

What are the issues, with respect to the client/server environment, a buyer or developer should consider when choosing a DBMS? As mentioned earlier, interoperability is very important if the system will live in a heterogeneous world. There is good news for buyers in this area. Vendors are finally coming together to define a standard for interoperability. The SQL Access Group is generating a standard interface which will provide interoperability via SQL. It should also be noted that Information Builders Inc.'s EDA/SQL product advertises the ability to link different DBMS platforms in a heterogeneous environment. As distributed database technology becomes more widespread, the level of interoperability will grow. Even if a particular vendor does not provide a complete solution today, the prudent buyer will judge the vendor's commitment to interoperability before making a purchase.

Another consideration for the buyer or developer is the type of database to choose. The relational model offers the best feature set for client/server computing. The older hierarchical and network databases offer one record-at-a-time access to data. This means a client requesting all records which match a particular selection criteria must make multiple requests of the server, and each request is answered by a single record. The single record approach significantly increases network traffic and the processing load on both client and server. The relational model, on the other hand, operates on *sets* of data rather than on individual records. A single request made of a server using standard SQL can result in a single reply from the server which contains all records which match the selection criteria. The reduced network load should be obvious. Customers in the process of migrating from host-based to client/server computing can use a SQL front-end on the server to access data in a non-relational DBMS. HP's ALLBASE/TurboCONNECT provides for read-only SQL access to data in a TurboIMAGE database.

Users with non-relational DBMS platforms that cannot use a SQL front end have another alternative. A custom server process could be written, using a variety of network application programming interfaces (API), that receives a request from the client, performs the appropriate processing on the server, and returns a reply to the client. Such a mechanism is also appropriate for very specialized processing where commercially available applications or toolsets are inadequate for the particular task. Information on available protocols and APIs is in a following section.

While the set orientation of SQL represents a non-trivial level of intelligence on the server, still higher levels provide real benefits to users. Features known as stored procedures and triggers provide additional intelligence. An example of these capabilities should make clear their value.

Imagine a client/server application which processes rather complex transactions. A transaction may consist of two table insertions, three deletes and six updates. Adding transaction delimiters, the total number of database requests made by the client of the server is thirteen. Now suppose this transaction can be "compiled" and stored on the server so that it can be called with a single request that includes as parameters all of the data values required by this stored procedure. One somewhat larger request has taken the place of *thirteen* separate requests. Again, the reduction in network utilization should be clear. Stored procedures also help insure data integrity. While any good DBMS will support the concept of atomic transactions by rolling back incomplete transactions, stored procedures reduce the likelihood a transaction will be incomplete. Because client platforms exist in the rough-and-tumble world of the user, many unpleasant things can happen to a client machine in the midst of issuing a transaction. Power failures, "operator difficulty" and the instability of the DOS operating system on PCs are just a few that come to mind. Issuing a transaction by making one call to the server instead of several increases the chance the transaction will complete.

A trigger is mechanism by which a DBMS will perform some function internal to the DBMS when a particular situation arises. Calculating a reorder point in an order processing system is a good example. Without using a trigger, a client would have to compare the quantity-on-hand with the reorder quantity each time the client requests a product. By implementing the function as a trigger within the DBMS, the database uses its own intelligence to perform the calculation. Thus performance and integrity of the system are increased.

To summarize, the developer or buyer should keep in mind the following criteria for the server DBMS:

Choose a relational DBMS or use an available or custom front-end to a non-relational DBMS.
Make sure the database vendor is committed to interoperability using industry standards.
Utilize stored procedures and triggers to increase the intelligence of the server, thereby creating higher performing and more reliable applications.

Networking Infrastructure

The network is a fundamental component of distributed computing using the client/server model. Specifically, the Local Area Network (or a connected group of LANs known as an *internetwork*) makes it possible to create a better application by distributing pieces of it between clients and servers. Networking is a broad and rapidly changing subject. Fortunately, the developer or buyer of client/server systems must be concerned with only a portion of the technology.

Many readers are probably familiar with the Open Systems Interconnection model as defined by the International Organization for Standardization (Fig. 2). The Physical and Data Link layers of the protocol stack, where technologies like Ethernet and Token Ring are defined, are of little interest to the client/server architect. At the Transport layer, and to some degree the Network layer, the developer begins to have control over how data is placed on the network. It is at these layers that common transports such as TCP/IP and Novell's SPX/IPX are defined. At even higher layers,

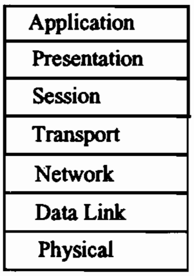


Figure 2

the architect has direct control over the network interface. This control is commonly implemented via an API, or Application Programming Interface.

A number of APIs are available to the developer. The most popular one in the PC environment has been NetBIOS. Novell has a large share of the PC market with SPX/IPX. OS/2 and LAN Manager introduced Named Pipes. Berkeley Software Distribution (BSD) introduced a socket interface, now commonly known as Berkeley sockets, on UNIX that has become a part of the TCP/IP protocol suite. Two good questions can be asked regarding these different APIs: As a developer, what are the differences in the APIs? As a buyer, is any of this important? While a complete discussion of network APIs is beyond the scope of this paper, a summary comparison of APIs is necessary.

The problem the developer faces is that once a substantial effort is made to create a product based on a particular API, the developer may then be "locked in" to that API. Obviously it is important to choose the best possible API at the beginning of a project. APIs can be compared based on the following factors: popularity, internetwork support, degree of openness, performance, and outlook for the future. The following table provides a summary of such a comparison.[4]

API	**Internet support**	**Degree of openness**	**Performance**	**Outlook**
NetBIOS	No	Primarily PCs	Good	Fading
SPX/IPX	Yes	NetWare (portable/native)	Excellent[1]	Remaining popular
Named Pipes[2]	Yes	Good	Good	Good
BSD Sockets	Yes	UNIX & others	Good	High

[1] Low overhead and efficient with small packets

[2] Named Pipes is a transport-independent API. If the underlying transport service is routable, then the API supports internetworking. Currently available only with LAN Manager implementations, including LM/X and LM/XL.

One of the technologies that underlies the client/server model is the concept of the Remote Procedure Call, or RPC. An RPC is simply a mechanism for a process executing on one system to call a procedure on another. The advantage of using an RPC is that they tend to be independent of the network transport. This means

[4] Safe, Scott, "Developing Client-Server Applications in HP System Environments using Industry Standard API's", *Proceedings of the 1991 INTEREX HP Users Conference*, August, 1991, Paper #2031

a developer can use a higher level mechanism to write an application and at the same time reduce the chance of being "locked in" to a particular network transport. As noted in the table above, Named Pipes is one API that provides a transport-independent interface. The Network Computing System, an RPC developed by Hewlett-Packard, is another.[5] While neither of these are currently supported on all network transports, their popularity should increase their breadth of implementation. NCS has the additional advantage of being part of the Open Software Foundation's Distributed Computing Environment specification. The implications of OSF DCE will be discussed shortly.

Network APIs and RPCs are not of major concern to the buyer of client/server systems. The buyer should ensure the supplier provides a solution supporting the network transport the buyer may already be using and that the supplier is committed to network transparent implementations in the future, preferably using the DCE specification.

Personnel Requirements

> *"There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things."* - Niccolo Machiavelli, *The Prince.*

Perhaps associating Machiavelli's thoughts on implementing change with client/server development over-states the difficulty of such development, but it does serve as a reminder that preparation is essential. Client/server computing presents new challenges to the information systems staff at every stage, from analysis to implementation. Not only may the concepts be new to many, but the model may necessitate the learning of new development tools. The traditional COBOL programmer may not be familiar with the concept of a procedure call with parameters, much less the additional complexity of the Remote Procedure Call. Preparation and adequate guidance are critical to success.

Every software supplier or in-house development organization should have a program in place to ensure their staff is capable of creating quality applications. Appropriate training is a must. An independent consultant can be extremely valuable in guiding the developer or purchaser through implementation and by providing supplementary technical assistance. Remember that the cost of training and assistance can be amortized over other projects; the cost of failure is payable in-full upon defeat.

---

[5] Lyons, Tom, *Network Computing System Tutorial*, (Prentice-Hall, 1991), p. 318

Implications of OSF's Distributed Computing Environment

The Open Software Foundation, in its Distributed Computing Environment, provides "enabling technologies" to producers of software to help those producers generate products that work together in a heterogeneous, networked environment.[6] The enabling technologies are not products themselves, but a set of services, delivered as source code, that vendors may use to generate open, interoperable products. The capabilities designed into the DCE include: Ease of use, Ease of administration, Security, Performance, Availability and Scalability.[7] A summary of the services provided by DCE appear in the following table.[8]

Service	Description
RPC	Network transparent RPC mechanism with security
Distributed Naming	Allows users and applications to identify network resources by name regardless of location
Time	Multi-system clock syncronization
Threads	Portable facility to create individually dispatchable threads of execution that share a common address space
Security	Authentication and authorization service based on Kerberos
Distributed File System	Creates a virtual global file system from a number of remote file systems

So why is DCE important to the developer or buyer of client/server systems? Because interoperability between disparate clients and servers is tremendously enhanced when both conform to DCE specifications. The DCE represents a complete toolset for creating client/server solutions. Although the enabling technologies are not available yet on all platforms, developers and buyers should look forward to the time when they are available. Users will benefit by receiving a virtual guarantee of interoperability: developers can spend resources solving customer problems instead of generating interfaces to an endless number of systems. The net result is lower cost applications that better satisfy customer requirements.

[6] *Interoperability: A Key Criterion for Open Systems*, Open Software Foundation , November, 1991

[7] *OSF Distributed Computing Environment Rationale*, Open Software Foundation, May, 1990, pp. 4-5

[8] *Distributed Computing Environment*, Open Software Foundation, November, 1990, pp. 3-10

Summary

Client/server computing is a big step toward user-driven information processing. Users, however, can still only specify requirements; they cannot be expected to be familiar with the technological details. The information systems professional can, by understanding both the forces behind the client/server model and the technologies required to make it work, help the user be successful. In the new age of computing, successful users create successful IS professionals.

Paper 3852

Stored Procedures and Rules in ALLBASE/SQL

Shu-Feng Wei, Carol Ann Krug, Amelia Carlson

Hewlett-Packard Company

19111 Pruneridge Avenue

Cupertino CA 95014

Tel No.: (408)447-6785 Fax No.: (408)447-0872

Abstract

ALLBASE/SQL introduces two important productivity and performance enhancements. Stored procedures provide a mechanism of collecting SQL statements and flow control into a single executable unit, and rules provide a method of placing triggers on data modification operations.

A stored procedure is a collection of SQL, conditional, and control statements, and local variables. Parameters are used to pass information to and from a procedure. Stored procedures are directly executable and sharable by users or applications, and are invoked by rules to perform the triggered actions.

A rule is automatically fired by a data modification statement on the monitored table, when a specified condition is met. The rule invokes a stored procedure to perform actions. Chains of rules provide a powerful mechanism for enforcing general constraints, general business rules, security checks, and audit control.

Stored procedures and rules localize more complex operations and constraints in the database. Business rule information that formerly resided in applications may now reside in the database, thus improving performance, reducing network traffic, enhancing database integrity, and facilitating application maintenance.

This paper describes the usability and functionality of these features, and demonstrates their benefits. This information will be of interest to database designers and administrators, and application developers.

Introduction

As organizations move toward decentralized operations to enhance their competitiveness, the client/server architecture has emerged as a solution to the integrity, resource management, security, and performance requirements of this enterprise-wide environment. In response to this business trend, ALLBASE/SQL now introduces organization-wide parameterized queries, transactions, and business rules to be programmed into the SQL client/server environment. ALLBASE/SQL supports the creation of database objects known as stored procedures and rules, which provide for a high degree of data consistency and integrity inside the DBEnvironment without the need for extensive application programming. Stored procedures define sequences of SQL statements and flow control that can be stored in the DBEnvironment and applied as a group either through rules or through execution by specific users. The concept of a stored procedure is not bound to the programming language environment but to the SQL server environment. Rules define complex relationships among tables by tying specific procedures to particular kinds of data manipulation on tables.

Stored procedures and rules offer the following significant benefits in the areas of performance, application design, security, integrity, and interoperability:

- Improve performance and minimize network traffic by combining conditional and control statements, and multiple SQL statements, into a database object that is stored and executed as a unit.
- Enforce general integrity constraints, general business rules, security checks, and audit control with a combination of procedures and rules, which may be chained.
- Reduce application code development and maintenance time with reusable and sharable procedures.
- Insulate applications from changes in the database schema by accessing the DBEnvironment only through procedures. Applications need not be modified nor recompiled unless the procedure parameters must also be changed.
- Provide additional security by controlling exactly which operations users can perform on database objects. Users granted EXECUTE authority on a procedure need not be granted direct authorization on any of the database objects accessed by the procedure.
- Enhance site autonomy in a distributed environment by allowing modification only through procedures.

Example Schema

The examples in this paper show stored procedures that access, and rules that are defined on, tables from the ALLBASE/SQL sample database, which comes with the ALLBASE/SQL product. A description of the tables and their columns follows:

- PurchDB.Parts (PartNumber, PartName, SalesPrice)
- PurchDB.Inventory (PartNumber, BinNumber, QtyOnHand, LastCountDate, CountCycle, AdjustmentQty, ReorderQty, ReorderPoint)
- PurchDB.SupplyPrice (PartNumber, VendorNumber, VendPartNumber, UnitPrice, DeliveryDays, DiscountQty)
- PurchDB.Vendors (VendorNumber, VendorName, ContactName, PhoneNumber, VendorStreet, VendorCity, VendorState, VendorZipCode, VendorRemarks)

- PurchDB.OrderItems (OrderNumber, ItemNumber, VendPartNumber, PurchasePrice, OrderQty, ItemDueDate, ReceivedQty)
- ManufDB.TestData (BatchStamp, TestDate, TestStart, TestEnd, LabTime, PassQty, TestQty)
- ManufDB.SupplyBatches (VendPartNumber, BatchStamp, MinPassRate)

Procedures

Procedure Features

An ALLBASE/SQL procedure is a collection of SQL statements, conditional and control statements. the user's request or through the firing of a rule. Internally, a procedure consists of a collection of sections (one section each for most SQL statements), plus internal instructions to implement the logic of the procedure and the non-SQL statements.

Parameter values may be passed to and from a procedure. Within a procedure, it is possible to declare local variables, assign values to parameters and local variables, issue most SQL statements, create looping and branching structures, test for error and warning conditions, and print messages. Status information may also be returned to the caller.

Parameters. A parameter represents a value that is passed between a procedure and an invoking application or rule. A parameter may be of any SQL data type, except the LONG data types, and may also be of type TID. Default values, nullability, and language may be defined just as for columns in a CREATE TABLE statement. All parameters to a procedure are input parameters. A parameter may also be designated as an OUTPUT parameter.

In the example in Figure 1-1, PartNumber, PartName, and InputPrice are formal parameters to the procedure NewPrice. If the part number is in the *Parts* table, the price is updated, and the part name is returned to the calling application program. Therefore, the PartName parameter is specified for OUTPUT in both the CREATE PROCEDURE and EXECUTE PROCEDURE statements. Within the body of the NewPrice procedure, parameter names are prefixed with a colon(:), and may be assigned a value or used in an expression.

Local Variables. A local variable holds a data value within a procedure. Local variable declarations must appear at the beginning of the main procedure body. A local variable may be of any SQL data type, except the LONG data types, and may also be of type TID. Default values, nullability, and language may be defined just as for columns in a CREATE TABLE statement. A local variable name may not duplicate a parameter name. Local variables function in procedures much as host variables function in application programs, but the two are not interchangeable. Within application programs, values contained in host variables can be passed as parameters in the EXECUTE PROCEDURE statement, but the application's host variables cannot be directly accessed from within the procedure.

In the NewPrice procedure in Figure 1-1, the local variable SalesPrice is used in the INTO clause of the SELECT statement, and in the RemovePart procedure in Figure 1-4, VendPartNum and BatchStamp are used in the INTO clause of the FETCH statement.

The internal representation of a nullable parameter or local variable includes a null indicator. Therefore, local variables and parameters do not require separate indicator variables. The values of nullable variables and parameters may be tested directly with the IS [NOT] NULL predicate. In the NewPrice procedure, the IS NULL predicate is used to test if the parameter InputPrice is NULL.

Built-in Variables. Built-in variables provide status information for the SQL statement most recently executed by the procedure, for use in runtime status checking and error handling. These built-in variables serve a similar function as the SQL Communication Area in an application program. The following built-in variables are provided:

::sqlcode	Error number, or end of scan indicator
::sqlerrd2	Number of rows operated on
::sqlwarn0	Warning indicator
::sqlwarn1	Character string truncation indicator
::sqlwarn2	Aggregate function argument null value indicator
::sqlwarn6	Transaction rollback indicator
::activexact	Active transaction indicator

The built-in variables are read-only, and are available only within procedures. The built-in variables are assigned their values upon completion of the execution of an SQL statement. These values persist until another SQL statement is executed by the procedure. That is, the values of the built-in variables are *not* affected by the execution of conditional and control statements, assignment statements, or print statements.

The NewPrice procedure includes a test for an *::sqlcode* value of zero to determine if the first SELECT statement executed without errors, just as in an application program. Similarly, a test for an *::sqlcode* value of 100 is used to determine if no tuple qualifies, and a test for an *::sqlerrd2* value of 1 is used to determine if a tuple was returned.

Conditional and Control statements. Procedures allow the testing of boolean expressions, and the creation of branching and looping structures, using the IF, WHILE, GOTO and RETURN statements.

IF statement

The IF ... THEN ... ENDIF statement and related clauses are used to allow conditional execution of one or more statements within a procedure. The condition expression is similar to an SQL search condition, and may include references to parameters, local variables, and built-in variables.

WHILE statement

The WHILE ... DO ... ENDWHILE statement is used to allow looping within a procedure. The condition expression follows the same rules as described above for the IF statement condition. This statement is very useful when fetching and manipulating multiple rows using a cursor (Cursors may be declared and manipulated in a procedure just as in an application).

Jump statements (GOTO, RETURN)

The GOTO statement permits a jump to a labeled statement within a procedure. The RETURN statement causes the execution of the procedure to halt, and returns control to the invoking user, application program, or rule. An optional integer return status may be included in the RETURN statement. The meanings of any return status values are completely user-defined. The NewPrice and ReceiveItem procedures return non-zero status codes to the caller to indicate the specific error encountered. The NewPrice procedure is called using a *ReturnStatus Variable* named *rtnstatus*. On returning from the procedure, if the *sqlcode* value in the SQL Communication Area indicates that the procedure was executed (i.e., *sqlcode* = 0), the value of *rtnstatus* may be tested to determine whether any error occurred in a procedure statement.

PRINT statement. The PRINT statement is used inside a procedure to pass informational messages back to the caller. The PRINT statement stores the content of a user-defined string, local variable, parameter, or built-in variable in the message buffer for display on return to the caller. On returning to the application, SQLEXPLAIN may be used in a loop to extract all the messages generated by PRINT during the operation of the procedure.

SQL statements. Most SQL statements that are allowed in embedded SQL application programs may also be used in procedures. Exceptions are BULK statements, statements connecting or releasing the DBEnvironment, preprocessor-specific directives, dynamic queries, and EXECUTE statements. Also, nested procedure creation and execution are not allowed. In procedures invoked by rules, transaction management statements, such as BEGIN WORK, COMMIT WORK, ROLLBACK WORK, and SAVEPOINT, are not allowed. Within a procedure, parameters or local variables must be specified to handle single-row query results. If more than one row qualifies for the query result, only the first row is returned into the parameters or local variables specified in the INTO clause, and a warning is issued.

The NewPrice procedure updates prices or adds new parts to the PurchDB.Parts table depending on whether the part number is currently in the table. The parameter PartNumber supplies a value for the predicate in the SELECT statement and later supplies a value for the VALUES clause in the INSERT statement and for the SET clause in the UPDATE statement. The procedure tests the built-in variable *::sqlcode* for a value of zero to ensure that the SELECT was successful before updating or inserting data into the PurchDB.Parts table.

For multiple row query results, a cursor may be used to loop through the result set and perform desired operations on one row at a time. The RemovePart procedure uses two cursors to scan the PurchDB.SupplyPrice and ManufDB.SupplyBatches tables for entries that correspond to a deleted part number. The procedure then performs deletions of qualifying rows in PurchDB.OrderItems and ManufDB.TestData.

Procedure Use

Procedures may be executed directly using an EXECUTE PROCEDURE statement. Procedures may also be invoked through the firing of a rule. When the rule is fired, the specified procedure is executed once for each qualifying row that satisfies the firing condition of the rule.

Passing Parameter Values. Input parameters can be passed to a procedure using host variables or literal values. Host variables can only be passed through application programs. For any OUTPUT parameter, a single host variable must be used. For a non-OUTPUT parameter, the value expression may include anything valid in an SQL expression except a subquery, an aggregate function, or a LONG, string, or TID function. Parameter values may be passed as either an ordered, unnamed list, or a named, unordered list. The example in Figure 1-1 shows the definition of the NewPrice procedure, followed by an excerpt from an application program that executes the procedure. Host variables are used to pass in the values of the PartNumber, PartName, and InputPrice parameters. Each host variable in the EXECUTE PROCEDURE statement maps in sequential order to a parameter definition in the CREATE PROCEDURE statement. Inside the procedure, values are referenced by parameter name, not by host variable name. In the ReceiveItem procedure in Figure 1-2, an indicator variable is used in the parameter list of the EXECUTE PROCEDURE statement to indicate a null value. But inside the procedure, the parameter itself contains the null value. In order to pass a null DateStamp to the procedure, the indicator variable DateStampInd is set to -1 and the content of the host variable DateStamp is undefined. Inside the procedure, the parameter DateStamp has the value NULL, and this fact can be determined through the ":DateStamp IS NOT NULL" test.

Returning Output Parameters. Data values may be returned to an application from a procedure by using the OUTPUT option in the parameter list. In the NewPrice procedure, the OUTPUT option is used in both the CREATE PROCEDURE and EXECUTE PROCEDURE statements.

Returning a Status Code. A return status code is an integer value returned from a procedure not invoked by a rule. The meaning of the code is defined within the procedure and may be tested on returning from procedure. The return status code may be used to indicate the success or failure of the procedure, the value of *sqlcode*, or any other piece of information that can be represented by an integer. In the execution of the NewPrice procedure, the integer host variable rtnstatus is used to hold the return status code. On return from the execution of the procedure, the value of *sqlcode* in the SQL Communication Area is tested first. Only if the value is zero, is the return status value tested. If *sqlcode* is not zero, the return status is undefined, since the procedure failed to execute properly. After testing *sqlcode*, and possibly the return status, any messages can be displayed by calling SQLEXPLAIN.

Error Handling. Explicit mechanisms for error handling must be provided inside procedures. By default, when an error occurs in an SQL statement in a procedure, the effects of the SQL statement are undone, but the procedure continues on to the next statement. The use of a WHENEVER SQLERROR directive with the STOP option will cause a rollback and an immediate return from the procedure on any error. If an error occurs in evaluating the condition in an IF or WHILE statement, or in evaluating the expression in a parameter or variable assignment statement, the execution of the procedure terminates, and control is returned to the caller with *sqlcode* set to the number of the error encountered.

SQLEXPLAIN may not be used within procedures. However, the built-in variable *::sqlcode* holds the error code from the first error message in the message buffer, which is guaranteed to be the most severe error. In procedures, as in ALLBASE/SQL applications, the message buffer is cleared out only before executing an SQL statement. As shown in the NewPrice procedure, *::sqlcode* must be tested before the next SQL statement is executed, or the information is lost. The RAISE ERROR statement may be used to generate an error within a procedure and make a message available to callers.

Upon return from an EXECUTE PROCEDURE call, the value of *sqlcode* indicates the success of procedure execution itself, not of any individual statement in the procedure. After returning from an EXECUTE PROCEDURE statement, an application should test *sqlcode* to determine if the procedure was executed, and *sqlwarn[0]* for possible errors, warnings, and informational messages.

A non-zero *sqlcode* is returned from procedure execution in the following situations:

- The procedure was not executed at all.
- An error occurred in the procedure when a WHENEVER SQLERROR STOP directive was active.
- An error occurred in evaluating an IF or WHILE condition or an expression in an assignment statement.
- An error occurred in copying output values to the user's host variables.

In all other cases, *sqlcode* is zero on return from a procedure, including cases in which one or more errors occurred in a procedure, but did not cause the procedure to stop. Messages for any errors from the last SQL statement executed by the procedure are available on return from the procedure by testing *sqlwarn[0]* for a value of 'W' and using SQLEXPLAIN. The execution of the NewPrice procedure in Figure 1-1 demonstrates this technique.

Validation and Revalidation. At run time, an application may execute a stored procedure, pass parameter values to the procedure, and may expect return values. If any object referenced in a query in the procedure is ever modified, the affected procedure section is marked invalid. The next time the procedure executes this query, the section is automatically recompiled to incorporate the new information, and silently generate a new, valid plan. Procedure sections may also be explicitly revalidated using the VALIDATE command. Invalidation and revalidation work for procedure sections just as for module sections, based on dependencies stored in the system catalog.

The procedure instructions can only become invalid due to internal changes in a new ALLBASE release. When migrating forward or backward to a new release, SQLMigrate marks each procedure as invalid. When an invalid procedure is executed, the procedure instructions and sections are silently regenerated. This process may also be performed explicitly using the VALIDATE command.

Authorization. Procedure creation requires RESOURCE or DBA authority, as well as appropriate authorities on all objects referenced by the procedure. However, direct execution of a procedure requires only EXECUTE authority on the procedure. Similarly, invocation of a procedure by a rule requires only the authority to execute the original statement that fires the rule.

Recommendations

The use of procedures can have indirect consequences that the procedure writer and the procedure caller may not anticipate. Problems are most likely to arise in the areas of transaction management, cursor management, error handling, and DBEnvironment settings. Procedure specifications should be well documented. In order to minimize difficulty, good communication between the procedure writer and the caller is essential.

The following practices are suggested for the procedure caller to ensure that a procedure is always called under the same conditions and with the same expectations:

- If the procedure may execute transaction statements, the application should not call the procedure with any work pending. Any work done by the application should be committed or rolled back before calling the procedure.
- If the procedure may execute transaction statements, the application should not call the procedure with any open cursors. Any cursors opened in the application with the KEEP cursor option and subsequently committed should be closed and committed before the application calls the procedure.

The following practices are suggested for procedure writers to ensure that a procedure will always execute as expected:

- Procedure execution should not span transaction boundaries. If the procedure executes any transaction statements, it should not terminate with any work pending. Any work done by the procedure should be either committed or rolled back before procedure termination.
- Cursors should not be kept open across procedure boundaries. If the procedure uses any cursors, they should be closed before termination.

- Procedures should not change the isolation level or other aspects of the DBEnvironment session without restoring them before termination.
- A procedure should not include any data definition statement (such as DROP PROCEDURE, TRANSFER OWNERSHIP OF PROCEDURE, REVOKE EXECUTE) that could invalidate procedure execution.

Examples

The code segment in Figure 1-1 shows a data entry application which either updates the price of a part, or adds a new part to the Parts table in the sample DBEnvironment PartsDBE, depending on whether the part number is currently in the table.

```
EXEC SQL
CREATE PROCEDURE PurchDB.NewPrice (PartNumber CHAR (16) NOT NULL,
                                   PartName CHAR (30) DEFAULT ' ' OUTPUT,
                                   InputPrice DECIMAL(10,2)) AS
BEGIN
DECLARE SalesPrice DECIMAL(10,2);
WHENEVER SQLERROR GOTO errorhandler;
SELECT SalesPrice INTO :SalesPrice
  FROM PurchDB.Parts
  WHERE PartNumber = :PartNumber;

IF ::sqlcode = 0 AND
   ::sqlerrd2 = 1 THEN   -- Found a tuple
  IF (:Inputprice IS NOT NULL) AND
     (:SalesPrice <> :Inputprice) THEN
    UPDATE PurchDB.Parts SET SalesPrice = :InputPrice
      WHERE PartNumber = :PartNumber;
    IF ::sqlcode = 0 THEN
      PRINT 'UPDATE successful';
    ENDIF;
  ENDIF;
ELSEIF ::sqlcode = 100 THEN  -- Tuple not found
  INSERT INTO PurchDB.Parts (PartNumber, PartName, SalesPrice)
    VALUES (:PartNumber, :PartName, :InputPrice);
  IF ::sqlcode = 0 THEN
    PRINT 'INSERT successful';
  ENDIF;
ENDIF;
RETURN ::sqlcode;
errorhandler:
IF (::sqlcode = -14024) OR
   (::sqlcode = -4008) THEN      -- Deadlock or no memory
  RETURN 1;
ELSEIF (::sqlwarn6 = 'W') THEN  -- Transaction abort
  RETURN 2;
ELSE                                       -- Other error
  RETURN 3;
ENDIF;
```

```
END;

(* Execute the NewPrice procedure with :PartNumber in an application *)
EXEC SQL
  EXECUTE PROCEDURE :rtnstatus = NewPrice
    (:PartNumber, :PartName OUTPUT, :InputPrice);

(* Test return status only if sqlcode is zero *)
if sqlca.sqlcode = 0 then
   case :rtnstatus of
     0:  (* Successful procedure execution ... *)
     1:  (* Re-try the procedure execution ... *)
     2:  (* Terminate the application ... *)
     3:  (* Just report the error ... *)
   otherwise  (* Just report the error ... *)
   end;

(* Display any error, warning, or PRINT messages *)
while (sqlca.sqlcode <> 0) or (sqlca.sqlwarn[0] = 'W') do
   begin
   EXEC SQL SQLEXPLAIN :sqlmessage;
   writeln(sqlmessage);
   end;
```

Figure 1-1. Procedure to Update a Price or Add a New Part

The example in Figure 1-2 shows a procedure that processes an order item receipt as a single transaction. The quantity received in the OrderItems tuple is updated, as is the total quantity on hand in the Inventory tuple. If the order has not yet been completely filled, and the due date has passed, a remark is written to the Vendors table to warn that this vendor may be unreliable. When the DBEnvironment is successfully updated, the transaction is committed, a status of zero is returned, and the current quantity on hand is returned as an output parameter. However, if either the OrderItems or the Inventory tuple is not found, a non-zero status is returned, and the transaction is rolled back.

This example illustrates the use of return status and output parameters for communication with the calling application, the use of the TID data type for local variables to improve performance, checking for null values of parameters and local variables, and recommended methods for error handling and transaction management.

```
CREATE PROCEDURE PurchDB.ReceiveItem
                (OrderNumber  INTEGER NOT NULL,
                 ItemNumber   INTEGER NOT NULL,
                 Qty          SMALLINT NOT NULL,
                 DateStamp    CHAR (8),
                 QtyOnHand    SMALLINT OUTPUT) AS
BEGIN

DECLARE PartTID,
        ItemTID         TID DEFAULT NULL;
```

```
DECLARE VendPartNumber  CHAR (16);
DECLARE VendorNumber    INTEGER;
DECLARE OrderQty        SMALLINT;
DECLARE ReceivedQty     SMALLINT;

WHENEVER SQLERROR STOP;

-- Retrieve OrderItems tuple
SELECT TID (), VendPartNumber, OrderQty, ReceivedQty
  INTO :ItemTID, :VendPartNumber, :OrderQty, :ReceivedQty
  FROM PurchDB.OrderItems
  WHERE OrderNumber = :OrderNumber
  AND ItemNumber = :ItemNumber;

IF :ItemTID IS NULL THEN   -- Order Item not found
  ROLLBACK WORK;
  RETURN -1;
ENDIF;

-- Retrieve Inventory tuple .
SELECT TID (PurchDB.Inventory), QtyOnHand, VendorNumber
  INTO :PartTID, :QtyOnHand, :VendorNumber
  FROM PurchDB.Inventory, PurchDB.SupplyPrice
  WHERE PurchDB.Inventory.PartNumber = PurchDB.SupplyPrice.PartNumber
  AND PurchDB.SupplyPrice.VendPartNumber = :VendPartNumber;

IF :PartTID IS NULL THEN   -- Part Number not found
  ROLLBACK WORK;
  RETURN -2;
ENDIF;

-- Calculate new ReceivedQty for OrderItems tuple
IF :ReceivedQty IS NOT NULL THEN
  :ReceivedQty = :ReceivedQty + :Qty;
ELSE
  :ReceivedQty = :Qty;
ENDIF;

-- Check if order is now completely filled
IF :DateStamp IS NOT NULL THEN
  IF (:ReceivedQty < :OrderQty) AND    -- Order not yet completely filled
     (:ItemDueDate < :DateStamp) THEN  -- Due date passed
    UPDATE PurchDB.Vendors             -- Add remarks to Vendors tuple
      SET VendorRemarks =
          'Order quantity not received by item due date'
        WHERE VendorNumber = :VendorNumber;
  ENDIF;
ENDIF;

-- Calculate new QtyOnHand for Inventory tuple
IF :QtyOnHand IS NOT NULL THEN
```

```
  :QtyOnHand = :QtyOnHand + :Qty;
ELSE
  :QtyOnHand = :Qty;
ENDIF;

-- Update Inventory tuple
UPDATE PurchDB.Inventory
  SET QtyOnHand = :QtyOnHand
  WHERE TID () = :PartTID;

-- Update OrderItems tuple
UPDATE PurchDB.OrderItems
  SET ReceivedQty = :ReceivedQty
  WHERE TID () = :ItemTID;

COMMIT WORK;
RETURN 0;
END;
```

In the application:

```
(* Any application cursors that have been OPENed and subsequently *)
(* COMMITted, should now be CLOSEd.                                *)

EXEC SQL COMMIT WORK;

EXEC SQL EXECUTE PROCEDURE :returnstatus =
                   PurchDB.ReceiveItem (:OrderNumber,
                                        :ItemNumber,
                                        :Qty,
                                        :DateStamp :DateStampInd,
                                        :QtyOnHand OUTPUT);


(* Test return status only if sqlcode is zero *)
if sqlca.sqlcode = 0 then
   case :returnstatus of
     0:  begin
         writeln ('DBEnvironment has been updated.');
         writeln ('Quantity on hand: ', QtyOnHand:1);
         end;
    -1:  writeln ('OrderItem not found.');
    -2:  writeln ('Part Number not found.');
   otherwise
   end;

(* Display any error, warning, or PRINT messages *)
while (sqlca.sqlcode <> 0) or (sqlca.sqlwarn[0] = 'W') do
   begin
   EXEC SQL SQLEXPLAIN :sqlmessage;
```

```
writeln(sqlmessage);
end;
```

Figure 1-2. Procedure to Process an Order Item Receipt

Rules

Rules can be used to enforce referential integrity constraints, general constraints (maintenance of derived data), and any other general purpose rules. Rules are more flexible than simple integrity constraints, enabling you to incorporate complex business rules into the structure of a DBEnvironment with minimal application programming. Rules may have side effects, while integrity constraints cannot. An integrity constraint must be checked on each data manipulation statement, but a rule is fired only for statements of the type specified in the rule definition. Furthermore, the rule action is executed only for each row that satisfies any firing condition specified in the rule definition. Rules are most useful for defining complex relationships that cannot be modeled using the existing check, unique, or referential constraints. ALLBASE rules depend on procedures, since procedures are used as the mechanism of rule enforcement.

Rule Features

Statement Type. Each rule is associated with a table, and is defined to fire upon each data manipulation statement of a specified type (insert, delete and/or update) applied to the table. For update statements, the rule may also be defined to fire only on update of specific columns. In the example in Figure 1-8, the CheckAuth2 rule is defined to fire only on statements that update the SalesPrice column of the Parts table.

Firing Condition. When a rule is defined, a firing condition may also be specified. Any row affected by the statement must also satisfy the firing condition in order to fire the rule. A firing condition is similar to a search condition in a query, and may include references to old and new column values of each row affected by the statement. The specification of a firing condition minimizes overhead by avoiding unnecessary procedure calls. For each row affected by the statement, the firing condition is tested, and the procedure is invoked only when the firing condition is satisfied. Thus, the specification of a firing condition can improve the performance of data manipulation operations on the table on which the rule is defined. In the example in Figure 1-8, the CheckAuth1 rule is defined to fire only on rows satisfying the condition:

```
OLD.SalesPrice > 3000.00 OR NEW.SalesPrice > 3000.00
```

Old and New Correlation Names. References to the old and new column values of each row affected by the statement may be included in the rule firing condition and in the parameter values passed to the procedure. The default correlation names are 'OLD' and 'NEW', and alternative correlation names may be specified in the REFERENCING clause. For an INSERT statement, an old correlation name will refer to the new values of the row, since no old values are available. Similarly, for a DELETE statement, a new correlation name will refer to the old values of the row, since no new values are available. In the example in Figure 1-7, the names 'Before' and 'After' are specified as the old and new correlation names in the LogUpdate rule, and the correlation names are used in the parameter values passed to the LogUpdate procedure.

Rule Use

Rule Action. The rule action may be anything that can be performed in a procedure (subject to some restrictions). That is, the action may include execution of most SQL statements, as well as conditional and control statements, and use of local variables and parameters. Typical uses of rules are to enforce complex integrity constraints, provide audit trails, or to enforce a user-defined authorization scheme. For example, to enforce a referential integrity constraint, a rule/procedure combination could be written to:

- Cascade an insert or delete of a foreign or primary key
- Disallow the deletion of a referenced primary key, or the insertion of a foreign key that does not match any primary key
- Nullify a foreign key that does not match any primary key

The figures in the Examples section below demonstrate each of these typical uses.

Any statement subject to a rule executes under statement level atomicity. Any error (SQL error or application-defined error) in the procedure invoked by the rule causes an immediate return from the procedure. Any work done by the procedure, as well as any effects of the statement that fired the rule, are undone. The RAISE ERROR statement may be used to generate an application-defined error.

Effects of Rules. Upon rule creation, any compiled module or procedure sections that reference the monitored table are marked as invalid. Thus, the first time such a section is executed or validated after rule creation, the section will be recompiled to include the rule information. At the time of rule creation, the rule is *not* applied to any existing rows in the table. A rule *only* affects data manipulation statements issued after the rule has been created, and when the rule is not disabled. Rules may be disabled, but not deferred. Any data manipulation operations that take place while a rule is disabled will never be subjected to that rule.

It is possible that more than one rule may apply to a single row affected by a data manipulation statement. In such cases, the order in which the rules will be applied cannot be guaranteed. If the order of rule application is important, the rules should be written so that their firing conditions are mutually exclusive.

Authorization. Rule creation requires OWNER authority on the table, and OWNER or EXECUTE authority on the procedure, or DBA authority. However, users issuing statements that cause a rule to fire need only the appropriate authority to issue the statement. No explicit authority on the rule or procedure is required.

Recommendations

A procedure invoked by a rule is somewhat specialized, and differs from a general procedure in the following important ways:

- No transaction statements are allowed, since the entire procedure execution must be part of a single transaction.
- Any error causes any work done by the procedure, as well as the original statement that fired the rule, to be undone.
- Since the procedure is executed in the context of a single tuple satisfying the rule firing conditions, the columns of that tuple are available, and may be passed as parameters to the procedure.

- Since procedure execution returns not to the calling application, but to the invoking rule, no return status or output parameters may be returned from a procedure invoked by a rule.

In addition to these restrictions, we make the following recommendations for rules, and procedures invoked by rules:

- Carefully analyze the relationships between rules, and procedures invoked by rules, to ensure that:
  - The maximum depth of rule chaining (20) is not exceeded
  - The performance implications on data manipulation statements that fire a chain of rules are well understood.
- To avoid an infinite loop of rule chaining, ensure that the procedure invoked by a rule does not itself contain any unconditionally executed statement that will fire the same rule.
- Ensure that multiple rules affecting the same data manipulation statement have mutually exclusive firing conditions, to avoid nondeterminism in the order of their application.
- In general, place conditions in the firing condition of a rule, rather than in conditional statements in the procedure invoked by the rule, to minimize the overhead of procedure execution.
- Avoid the use of the following statements, which alter transaction or session attributes, in procedures invoked by rules:
  - SET CONSTRAINTS
  - ENABLE/DISABLE RULES
  - SET PRINTRULES
  - SET USER TIMEOUT
- Avoid the use of data definition statements, especially CREATE RULE and DROP RULE statements, in procedures invoked by rules.

Examples

The examples in Figures 1-3 and 1-4 show two methods of cascading a delete of a primary key from the Parts table. Figure 1-3 shows a chained set of rules and procedures to remove all references to a part number once it has been deleted from the database. In this case a rule fires a procedure, which causes another delete, which causes another rule to invoke an additional procedure, and so on.

```
CREATE PROCEDURE PurchDB.RemovePart (PartNum char (16) not null) AS
  BEGIN
  DELETE FROM PurchDB.Inventory WHERE PartNumber = :PartNum;
  DELETE FROM PurchDB.SupplyPrice WHERE PartNumber = :PartNum;
  end;

CREATE RULE PurchDB.RemovePart
  AFTER DELETE FROM PurchDB.Parts
  EXECUTE PROCEDURE PurchDB.RemovePart (OLD.PartNumber);

CREATE PROCEDURE PurchDB.RemoveVendPart (VendPartNum char (16) not null) AS
```

```
      BEGIN
      DELETE FROM PurchDB.OrderItems WHERE VendPartNumber = :VendPartNum;
      DELETE FROM ManufDB.SupplyBatches WHERE VendPartNumber = :VendPartNum;
      END;

    CREATE RULE PurchDB.RemoveVendPart
      AFTER DELETE FROM PurchDB.SupplyPrice
      EXECUTE PROCEDURE PurchDB.RemoveVendPart (OLD.VendPartNumber);

    CREATE PROCEDURE ManufDB.RemoveBatchStamp (BatchStamp char(16) not null) AS
      BEGIN
      DELETE FROM ManufDB.TestData WHERE BatchStamp = :BatchStamp;
      END;

    CREATE RULE ManufDB.RemoveBatchStamp
      AFTER DELETE FROM ManufDB.SupplyBatches
      EXECUTE PROCEDURE ManufDB.RemoveBatchStamp (OLD.BatchStamp);
```

Figure 1-3. Chained Rules to Cascade a Delete of a Primary Key

Figure 1-4 uses a single rule and one procedure to remove all references to a part number once it has been deleted from the database. In this case, a single procedure RemovePart determines which rows need to be deleted in the other tables once a part number is deleted from the Parts table.

```
    CREATE PROCEDURE PurchDB.RemovePart (PartNum CHAR (16) NOT NULL) AS
      BEGIN
      DECLARE VendPartNum CHAR(16) NOT NULL;
      DECLARE BatchStamp DATETIME NOT NULL;
      DECLARE SupplyCursor CURSOR FOR
        SELECT VendPartNumber FROM PurchDB.SupplyPrice
          WHERE PartNumber = :PartNum;
      DECLARE BatchCursor CURSOR FOR
        SELECT BatchStamp FROM ManufDB.SupplyBatches
          WHERE VendPartNumber = :VendPartNum;

      DELETE FROM PurchDB.Inventory WHERE PartNumber = :PartNum;
      OPEN CURSOR SupplyCursor;
      FETCH SupplyCursor INTO :VendPartNum;
      WHILE ::SQLERRD2 = 1 DO
        DELETE FROM PurchDB.OrderItems WHERE VendPartNumber = :VendPartNum;
        OPEN CURSOR BatchCursor;
        FETCH BatchCursor INTO :BatchStamp;
        WHILE ::SQLERRD2 = 1 DO
          DELETE FROM ManufDB.TestData WHERE BatchStamp = :BatchStamp;
          FETCH BatchCursor INTO :BatchStamp;
        ENDWHILE;
        CLOSE CURSOR BatchCursor;
        DELETE FROM ManufDB.SupplyBatches WHERE VendPartNumber = :VendPartNum;
        FETCH SupplyCursor INTO :VendPartNum;
```

```
  ENDWHILE;
  CLOSE CURSOR SupplyCursor;
  DELETE FROM PurchDB.SupplyPrice WHERE PartNumber = :PartNum;
  END;

CREATE RULE PurchDB.RemovePart
  AFTER DELETE FROM PurchDB.Parts
  EXECUTE PROCEDURE PurchDB.RemovePart (OLD.PartNumber);
```

Figure 1-4. Single Rule to Cascade a Delete of a Primary Key

The choice of a single rule and procedure, or a set of chained rules and procedures, will depend on the task to be implemented.

The example in Figure 1-5 shows a rule/procedure combination that disallows the deletion of a primary key that is referenced in another table.

```
CREATE PROCEDURE PurchDB.CheckDelete (PartNum CHAR (16) NOT NULL) AS
BEGIN
DECLARE Count INTEGER;
SELECT COUNT (*) INTO :Count
  FROM PurchDB.Inventory
    WHERE PartNumber = :PartNum;
IF :Count > 0 THEN
  RAISE ERROR -7001 MESSAGE 'Foreign key value exists in Inventory table';
ENDIF;
SELECT COUNT (*) INTO :Count
  FROM PurchDB.SupplyPrice
    WHERE PartNumber = :PartNum;
IF :Count > 0 THEN
  RAISE ERROR -7002 MESSAGE 'Foreign key value exists in SupplyPrice table';
ENDIF;
END;

CREATE RULE PurchDB.CheckDelete
  AFTER DELETE FROM PurchDB.Parts
    EXECUTE PROCEDURE PurchDB.CheckDelete (OLD.PartNumber);
```

Figure 1-5. Rule to Disallow Deletion of a Referenced Primary Key

Finally, the example in Figure 1-6 shows a rule/procedure combination that nullifies a foreign key that no longer matches any primary key.

```
CREATE PROCEDURE PurchDB.Nullify (PartNum CHAR (16) NOT NULL) AS
BEGIN
UPDATE PurchDB.Inventory
  SET PartNumber = NULL
    WHERE PartNumber = :PartNum;
UPDATE PurchDB.SupplyPrice
```

```
    SET PartNumber = NULL
      WHERE PartNumber = :PartNum;
  END;

  CREATE RULE PurchDB.Nullify
    AFTER DELETE FROM PurchDB.Parts
      EXECUTE PROCEDURE PurchDB.Nullify (OLD.PartNumber);
```

Figure 1-6. Rule to Nullify a Foreign Key with no Matching Primary Key

Figure 1-7 shows a rule/procedure combination that provides an audit trail. In this example, any modification of the PurchDB.Inventory table is logged to a history table, along with the user name and a timestamp. This example also demonstrates the use of the OLD and NEW correlation names to refer to the old and new values of the row (before and after the INSERT, UPDATE, or DELETE statement is applied).

```
  CREATE PROCEDURE PurchDB.LogInsert (PartNumber INTEGER NOT NULL) AS
    BEGIN
    INSERT INTO PurchDB.InventoryHistory
      VALUES (NULL, :PartNumber, USER, CURRENT_DATETIME);
    END;

  CREATE RULE PurchDB.LogInsert
    AFTER INSERT INTO PurchDB.Inventory
    EXECUTE PROCEDURE PurchDB.LogInsert (NEW.PartNumber);

  CREATE PROCEDURE PurchDB.LogDelete (PartNumber INTEGER NOT NULL) AS
    BEGIN
    INSERT INTO PurchDB.InventoryHistory
      VALUES (:PartNumber, NULL, USER, CURRENT_DATETIME);
    END;

  CREATE RULE PurchDB.LogDelete
    AFTER DELETE FROM PurchDB.Inventory
    EXECUTE PROCEDURE PurchDB.LogDelete (OLD.PartNumber);

  CREATE PROCEDURE PurchDB.LogUpdate (OldPartNumber INTEGER NOT NULL,
                                      NewPartNumber INTEGER NOT NULL) AS
    BEGIN
    INSERT INTO PurchDB.InventoryHistory
      VALUES (:OldPartNumber, :NewPartNumber, USER, CURRENT_DATETIME);
    END;

  CREATE RULE PurchDB.LogUpdate
    AFTER UPDATE OF PurchDB.Inventory
    REFERENCING OLD AS Before NEW AS After
    EXECUTE PROCEDURE PurchDB.LogUpdate (Before.PartNumber, After.PartNumber);
```

Figure 1-7. Rules to Provide an Audit Trail

The rule/procedure combination in Figure 1-8 enforces a very simple user-defined authorization scheme. In this scheme, an authority level of 1 or higher is required to insert or delete a part with a salesprice greater than 3000.00. An authority level of 2 or higher is required to update the salesprice of a part if the difference between the old and new prices is greater than 1000.00. This example also demonstrates the use of the RAISE ERROR statement to generate an application-defined error, which causes the statement to be undone when the user does not have the required authority.

```
CREATE PROCEDURE PurchDB.CheckAuth (Level INTEGER NOT NULL) AS
  BEGIN
  DECLARE Ulevel INTEGER;
  SELECT Level INTO :Ulevel FROM PurchDB.UDAuth
    WHERE User = USER;
  IF (:Ulevel IS NULL) OR
     (:Ulevel < :Level) THEN
    RAISE ERROR -7003 MESSAGE 'User does not have the required authority';
  ENDIF;
  END;

CREATE кULE PurchDB.CheckAuth1
  AFTER INSERT, DELETE OF PurchDB.Parts
  WHERE OLD.SalesPrice > 3000.00
     OR NEW.SalesPrice > 3000.00
  EXECUTE PROCEDURE PurchDB.CheckAuth (1);

CREATE RULE PurchDB.CheckAuth2
  AFTER UPDATE (SalesPrice) OF PurchDB.Parts
  WHERE NEW.SalesPrice - OLD.SalesPrice > 1000.00
  OR    NEW.SalesPrice - OLD.SalesPrice < 1000.00
  EXECUTE PROCEDURE PurchDB.CheckAuth (2); ·
```

Figure 1-8. Rules to Enforce a User-Defined Authorization Scheme

Conclusions

The combination of stored procedures and rules in ALLBASE/SQL allows users to move complex operations and general constraints from their applications into the database environment itself, placing logic as well as data under DBMS control. Thus, entire transactions or sequences of statements, including conditional and control statements, may be stored as database objects, and executed with a single statement, or automatically through rules. The recommendations and examples described in this paper can provide a starting point from which users can work to develop procedures and rules for their own DBEnvironments. Use of procedures and rules can dramatically improve performance in a client/server environment by reducing network traffic, while enhancing database integrity, and facilitating application maintenance.

Paper Number: 3853

Improving the Bottom Line With Object-oriented Databases

Douglas Dedo

Hewlett-Packard
19111 Pruneridge Avenue 44MP
Cupertino, CA 95014
408-447-7726

ABSTRACT

There is intense pressure on large companies to reduce the time to market of their products, reduce costs, and increase their product and process quality. Object-oriented databases provide a powerful way to manage business processes and provide a decision support environment tailored to the way people work. There is also the potential of significantly reduced software maintenance costs, the most expensive phase of developing software. OpenODB from Hewlett-Packard is an example of how this technology can coexist with existing data and applications while providing users with an intuitive way to manage their most valuable business information.

One of the most challenging jobs these days is that of the Manager of Information Systems (MIS). Business factors have made their mark. The recent recession has forced tough financial decisions. Mergers, acquisitions, and industry regulations have resulted in unexpected changes. A common theme has been the need to do more with less. At the same time, technology has not always been the MIS director's friend. Low prices and decentralization have removed some of the ability to control one's future as individual departments make independent computer decisions. Local Area Networks and pools of data have proliferated. Current mission critical applications cost a lot to maintain and they cannot be cost effectively replaced. New technology must coexist with the legacy systems. Maintaining the infrastructure to tie an enterprise together has not been easy.

With this as the backdrop for information technology work in the 90's, what are the objectives for the next five years? A good business plan is as important for success today as using the right technologies. Technology advances that speed up software development, reduce software maintenance costs, and allow for end user access to all job-relevant data are sought by many companies today. This is based on the thinking that the company which gets new products and services to the market faster with a lower overall cost structure is in the best position to dominate that marketplace. This paper will describe a new type of tool called an object-oriented database management system (ODBMS) that portends business benefits in the commercial MIS environment.

What are objects?

Objects represent an end user's model of business information as an application building block in a computer. An object is defined by first classifying it as a certain type of business information. An example would be to have an "Employee" type of object defined to support a personnel application. For each type of object, a combination of data and associated functions are defined to accurately simulate the real world entity being modeled. For this Employee object type, the data might include a name, business phone, and salary. Functions that would be performed on Employees could include the hiring process or a mapping of the relationship between an employee and their manager.

Where is the biggest business benefit?

The most significant business benefit of object-orientation comes from its hierarchical structure. The underlying concept follows the structure of an animal kingdom chart. All attributes of mammals are inherited down through classes and phylum until we get to humans. Human beings inherit all mammal characteristics and then have some new ones as well. Within an object-oriented design, both the data and functions that are defined for one type of object are automatically inherited by object types lower down in the hierarchy. For example, the data and functions associated with an Employee would be applied to subtypes of Employees like "Engineers," "Managers," and "Contractors." The significance of this structure is that development work needs to be done in just one place. As well, when changes need to be applied, they are done in just one place. The technology automatically applies this work to each appropriate type of object. The resulting business benefit is a dramatically lower cost to maintain object-oriented applications.

The most expensive portion of the software development lifecycle is the maintenance phase. Companies have seen object-oriented projects reduce the cost of maintenance by a ratio of 3 to 1. That means that if an application developed with a structured programming approach required three people to support, then the same application with an object-oriented approach would need an average of one person. One example of this is demonstrated by work done at the Brooklyn Union Gas Company and documented by Gartner Group, an industry consulting company.* The object-oriented approach for this customer service application resulted in significantly less maintenance effort.

Reducing the cost of maintaining an application is a financial savings. But being able to redeploy two out of three developers onto new projects means an opportunity to get new services to customers faster as well. This can provide a powerful competitive advantage. This process is going to benefit commercial users of object-oriented technology more than code reuse that is often touted as the main benefit of an object-oriented approach.

Code reuse implies that code exists. In the commercial data processing world, with very few exceptions today, there is no object-oriented code to reuse. However,

* Burnstine, R. "Large-Scale OO Implementation: It Can Be Done!" Gartner Group, Software Engineering Strategies (Research Note SPA-700-422), and "Critical Factors for Success with OO" (Research Note E-700-421), June 18, 1990.

there is a lot of Cobol and to a lesser extent Fortran. New users of an object-oriented approach should expect the first project to take as long as projects using other design techniques. This is partly due to the need for a thorough object-oriented analysis and design at the beginning of a project. Once an object-oriented application is in production, the next project may not be able to reuse much of the first project's object code. This has to do more with how businesses work than to technology limitations. For example, this year a company will focus on some key areas for information technology solutions. Normally, when those are addressed, next year the focus will move to other functional areas within a company and the objects may well be different. But savings in maintenance can be achieved starting with the very first project.

Where is object-oriented technology being applied?

Object-oriented technology is being applied in four major areas: languages, environments, databases, and tools & methodologies. Object-oriented programming languages, like C++ and Smalltalk, are being used to craft the applications. These applications are being run in environments focused on different types of end users.

One type of environment designed for people who manage computer systems and networks is OpenView. OpenView was initially developed by HP and has been licensed by IBM and by the Open Software Foundation (OSF) for its Distributed Management Environment. Softbench is an environment for object-oriented programmers to integrate their choice of development tools. IBM has also licensed Softbench from HP as has Informix. NewWave has received a lot of visibility as an environment for business application users. American Airlines is one company that has made extensive use of NewWave and has also played an active role in the standards-oriented Object Management Group.

Object-oriented databases fit into the picture by safely storing objects, these combinations of data and code, so they can be shared among a group of related applications in the same way previous databases shared just data. For commercial applications, object databases should support multiple users simultaneously accessing the same information, a powerful query language to easily locate information from a vast warehouse of data, and data center quality security and integrity to prevent unauthorized access to sensitive information. Also, there is a need to model business processes that are facilitated by rollback capabilities and a need to maintain the stored data without disrupting mission critical applications with capabilities like online backups. Without an ODBMS, building object-oriented applications to support large numbers of users and large amounts of information becomes almost impossible.

Tools and methodologies complete the picture and relate to the other three areas. Tools can be used with object-oriented languages (e.g. debuggers), tools like a graphical user interface (GUI) builder can be integrated in Softbench, and graphical browser tools can be used to inspect the internal structure of a database. An emerging object oriented tool to tie each of these areas together is the Distributed Object Management Facility (DOMF). One of the DOMF services is an Object Request Broker (ORB) that makes the DOMF work like an object-oriented network backbone. Applications running in environments can make requests of the ORB for objects stored across a wide area network through the DOMF. The request is routed to a database where the object is stored.

There are a number of methodologies for designing object-oriented applications that tend to be extensions of the entity-relationship (ER) model commonly used with relational database design. It is possible to design an application this way, and then implement it using third generation languages and relational databases and still see business benefits of an object-oriented approach. In this fashion, it is possible to safely learn about objects while creating a structure that allows for object-oriented technology to be plugged in when it makes good business sense.

An Object Database Example

OpenODB, from Hewlett-Packard, is an object-oriented database that uses a relational database, Allbase/SQL, as its internal engine to store and manage the data portion of objects. It was determined that there is no difference in managing relational data from object-oriented data. Building OpenODB on top of a relational database meant reusing code that been enhanced over the last ten years. The object manager that sits on top just needed to focus on all of the object-oriented capabilities. This made it possible to bring a complete, commercial-quality ODBMS to the market much faster than if the system were built from the ground up (see Figure 1).

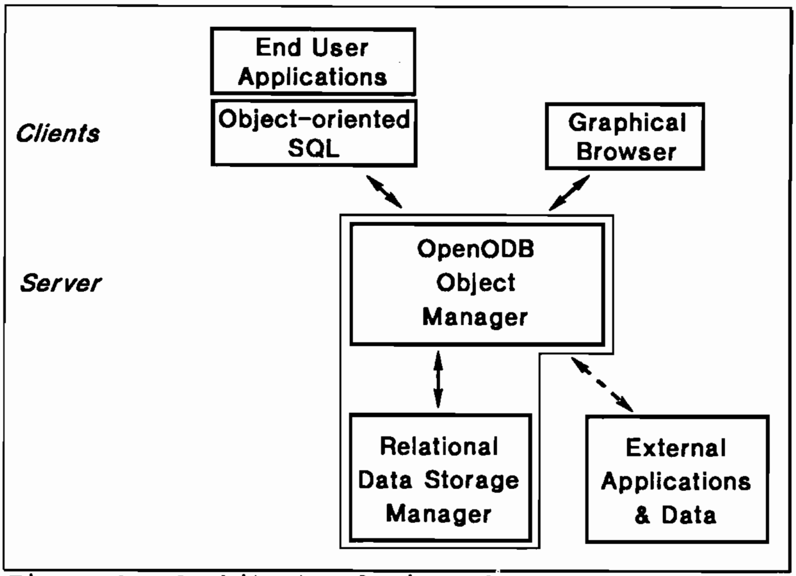


Figure 1. Architectural view of OpenODB.

The language used to create, manipulate, and query objects in OpenODB is called Object-oriented SQL (OSQL). As part of Hewlett-Packard's work in the ANSI and OMG standards activities for object databases, a paper was presented on OSQL to these groups and was well received. A query language is crucial when trying to locate one piece of information in a vast amount of data. OSQL can help developers locate object-oriented functions in the system (for reuse or updates) with commands that provide a way to inspect the structure of the database. The browser is a tool designed for developers to graphically display the internal structure of the database. This is another way to help locate available object-oriented functions and data.

OSQL can be called from any C-linkable programming languages including Cobol, Fortran, Ada, C++, and Smalltalk. This provides a way for developers to continue to use their favorite language or tools and access this new type of database. Object-oriented functions for OpenODB can be implemented with any combination of OSQL and C-linkable languages by using a mechanism called External Functions. Code that already exists in a company can be tied into the database in this way.

Foreign Functions can also be used to access any type of data located anywhere in a company's network. This flexibility was needed given that most commercial data is in many different formats. To use the Foreign Function interface, a developer would write code that goes across the corporate network and accesses the appropriate data. This code is then linked to an OSQL function. HP is working with Information Builders, Inc. to tie the Enterprise Data Access (EDA/SQL) tool together with OpenODB. EDA/SQL is a tool that accesses more than 50 commonly used databases over a number of networking protocols. This can lower the effort required to access enterprise-wide data.

What is different when using an ODBMS?

One of the best ways to understand how using an object database is different from other types of databases is to look at where the work is done and how it is done. Figure 2 shows how the work would be divided between an object-oriented application and OpenODB for a sample personnel example. Figure 3 shows a similar application that could be implemented using a relational database.

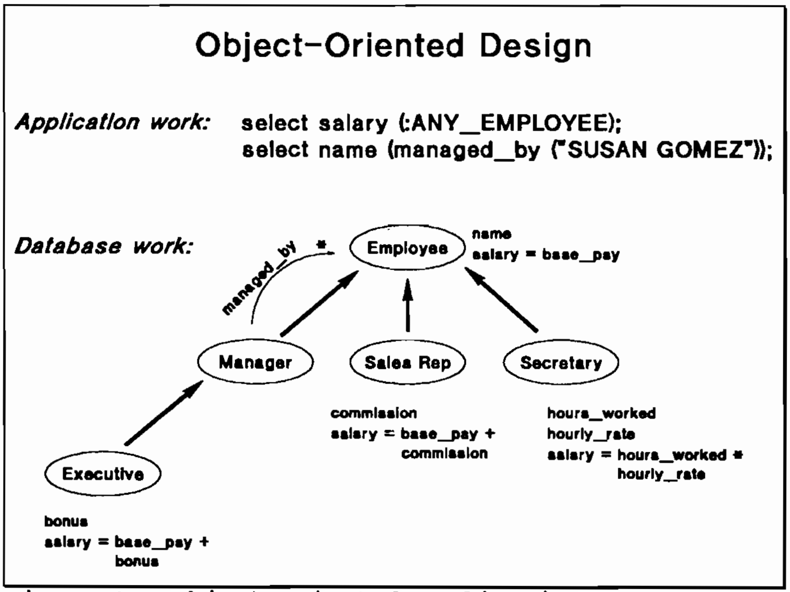
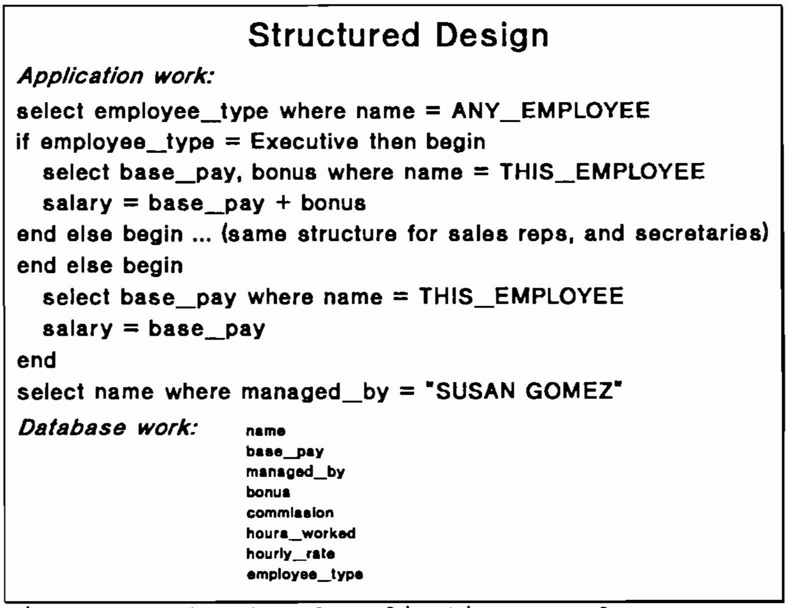


Figure 2. Object-oriented application example.

The object-oriented database design maps out an employee hierarchy. The name and salary functions are inherited by every subtype of employee. A relationship is created between managers and employees. The "*" signifies that a manager can manage more than one employee. For some of the subtypes of employee, like the Sales Rep, there is a need for additional data and for the salary function to be calculated differently.

When the salary function is called in the application, the developer does not need to know the type of the employee. The object database determines the type and automatically applies the appropriate salary calculation. Note that the same function name "salary" is used many times. Object databases provide a way to reuse the same function name with different object types representing how words are used in the real world. This is called function overloading or polymorphism. The complexity is reduced for application developers since the call in the object-oriented application remains the same.

The second application example, in both Figure 2 and 3, shows how to get the names of the people who work for Susan Gomez.



Structured Design

Application work:

```
select employee_type where name = ANY_EMPLOYEE
if employee_type = Executive then begin
  select base_pay, bonus where name = THIS_EMPLOYEE
  salary = base_pay + bonus
end else begin ... (same structure for sales reps, and secretaries)
end else begin
  select base_pay where name = THIS_EMPLOYEE
  salary = base_pay
end
select name where managed_by = "SUSAN GOMEZ"
```

Database work:

```
name
base_pay
managed_by
bonus
commission
hours_worked
hourly_rate
employee_type
```

Figure 3. Structured application example.

In the structured programming example, the data fields in the database may fit into a hierarchical, network-model, or table-oriented model. The application needs to first determine the type of employee and then apply the appropriate algorithm to get the salary. Implementation will vary between different databases, but none provide a way to use the same function name for different types of calculations or to inherit code across different types of employees.

These two examples show the power of inheritance and polymorphism. There are other variations on this example due to the complexity of a commercial environment. Consider the added complexity to the structured program when the data is in different types of databases on different types of computers. The object-oriented application would not need to change.

When a developer would like to reuse code in shared libraries today there is little automated help. Graphical browsers and OSQL are tools that can facilitate this search for code in the object database.

When the algorithms need to be modified due to changes in the tax law, for example, this means recompiling the structured application. Changes to the object database code can be done while the applications continue to support the business. Also a new type of employee, like a contractor, can be added without stopping the applications.

The last issue comes when working with spatial (three dimensional) data. One example would be to query what railway tracks are located within 5 miles of any airport in the country and highlight them on a map. OSQL is capable of supporting this type of relationship modeling and query. This type of data has been difficult for people to model in non-object-oriented databases.

Conclusion

Object-oriented databases provide a general purpose tool in which to model complex business environments. Through the use of the object-oriented model, it is possible to provide an environment in which the cost to develop and support applications can be reduced. This provides a way to speed up time to market of new services which can create a competitive advantage.

Some object databases, like OpenODB, can model business information and processes that can access any data available throughout a company's network. This can shield the complexity of an enterprise from application developers. However, an ODBMS is new technology and the learning curve as well as the object-oriented design will require a well planned approach before taking advantage of the associated benefits.

Douglas Dedo is the product marketing manager at Hewlett-Packard responsible for HP's object-oriented database program. Doug has held product marketing, marketing communications, and software development engineering positions within Hewlett-Packard's computer networking group. Previous to HP, he worked on networking and database projects as a software development engineer in the United States and Europe. Doug holds a B.A. degree in computer science from U.C. Berkeley.

HP 3000 Open Systems and Client/Server Computing

Christine Fronczak
Hewlett-Packard Company
Commercial Systems Division
(408) 447-7783

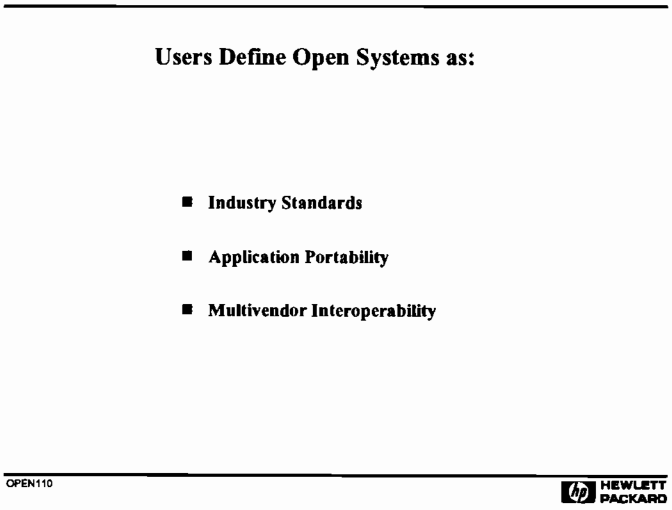
Table of Contents

INTRODUCTION

Hewlett Packard has a strong historical record of helping our customers protect their investments in information systems. In keeping with this belief, HP is committed to providing an open system on the HP 3000 through support of a wide range of industry standards. As a result of HP's broad systems offering and commitment to Open Systems, HP lays the foundation for development of Client/Server applications. In this paper, we will provide an overview of what it means in the industry today to be an "open system" and how we have developed open systems on the HP 3000. The major focus of this paper will be to discuss the features, services, and applications available on the HP 3000 for Client/Server computing.

I. OPEN SYSTEMS ON THE HP 3000

HP has embraced open systems as a strategic direction for the HP 3000 because HP sees value in bringing the strength of HP 3000 commercial processing to open systems. The HP 3000 open systems strategy is to combine the benefits of open and leading OLTP commercial functionality in a single offering. Existing HP 3000 customers can reap the benefits of open systems without a costly conversion to UNIX and new customers can obtain the value of HP 3000 commercial computing without giving up the benefits of open systems.



Users Define Open Systems as:

- **Industry Standards**
- **Application Portability**
- **Multivendor Interoperability**

OPEN110

HEWLETT PACKARD

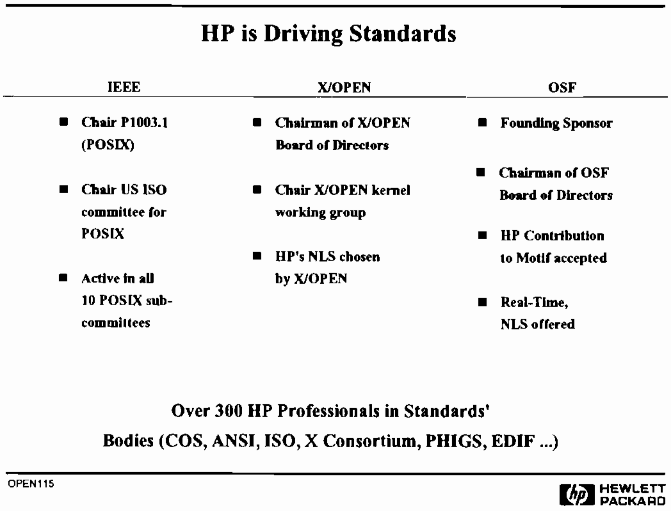
Let's first take a look at how Open is defined. Forrester Research and another independent survey of 2000 customers clearly indicate that customers seek two key benefits of open systems --interoperability and portability -- NOT any specific technology such as UNIX or POSIX.

In a 1990 market research study, HP tested the need for open systems and results showed a high need for the benefits of open systems. Users defined open systems according to a vendor's ability to meet support for:

1) Industry standards -- support for industry standards is what facilitates the key benefits derived from open systems such as multivendor interoperability and application portability.

2) Application portability -- the ability to provide hardware and software vendor independence for increased solutions flexibility and reduced software development costs. Application portability ensures that applications can coexist across different platforms.

3) Multivendor interoperability -- the ability to provide system networking integration with other systems including IBM, UNIX, PCs and workstations along with transparent access to information across the enterprise. Connectivity and coexistence with installed multivendor systems is the key benefit of multivendor interoperability.



HP is Driving Standards

IEEE	X/OPEN	OSF
■ **Chair P1003.1 (POSIX)**	■ **Chairman of X/OPEN Board of Directors**	■ **Founding Sponsor**
■ **Chair US ISO committee for POSIX**	■ **Chair X/OPEN kernel working group**	■ **Chairman of OSF Board of Directors**
■ **Active in all 10 POSIX sub-committees**	■ **HP's NLS chosen by X/OPEN**	■ **HP Contribution to Motif accepted**
		■ **Real-Time, NLS offered**

Over 300 HP Professionals in Standards' Bodies (COS, ANSI, ISO, X Consortium, PHIGS, EDIF ...)

OPEN115

HEWLETT PACKARD

INDUSTRY STANDARDS

The first area of functionality important to have for open systems is compliance with industry standards. HP not only complies with the key standards groups, but is driving industry standards with groups such as IEEE/POSIX, X/Open, and the Open Software Foundation.

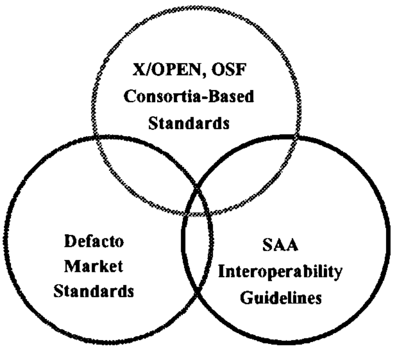
X/Open is an international consortium of computer vendors working to create an internationally supported, vendor neutral Common Applications Environment (CAE) based on de facto and industry standards. X/Open does not develop products but rather endorses international standards where they exist, and adopts de facto standards where they do not exist. X/Open efforts are embodied in a series of publicly available reference documents called the X/Open portability guide (XPG), which has become widely accepted as the dominant working definition of a full system environment. This standards group provides a set of practical unifying standards for users and vendors that bridges the gap between proprietary and industry standards, open systems and IBM's SAA.

Open Software Foundation (OSF) is a non-profit, industry supported research and development organization formed in 1988 by Hewlett Packard, Digital Equipment Corporation, Groupe Bull, IBM, Nixdorf, Phillips, Siemens, and Hitachi. More than 130 organizations are now members of OSF. Hewlett Packard is a key participant in OSF and currently chairs the OSF Board of Directors.

OSF functions as an integrator and implementor of technologies, building on existing standards and specifications and helping in the definition of new standards and specifications where none currently exist. OSF complements the work of standards groups such as X/Open and IEEE/POSIX by implementing the standards adopted or defined by these standards organizations. Already, OSF has accelerated the availability of several key technologies accessible by HP 3000 customers. One example is the OSF/MOTIF graphical user interface which builds on top of the X Window system standard.

HP has taken a clear leadership role to ensure the evolution of existing standards and the emergence of new standards through its innovations, such as NewWave's Object Management facility, OSF/MOTIF, and HP OpenView Network and Systems Management products.

Driving Forces Defining HP 3000 Standards



Support for relevant maturing commercial standards

OPEN116

HEWLETT PACKARD

Let's take a look at the driving forces that define the standards supported by the HP 3000 which encompass the commercial needs of the marketplace.

The first force driving the HP 3000 towards open systems is it's commitment to X/Open and OSF Consortia-based standards. The HP 3000 goal is to become XPG3 BASE branded in the future. This means supporting standards at the operating system and command level via POSIX, internationalization support (based on HP

Native Language Support), and the ANSI C language. As we have already discussed, HP is a very active participant on the X/Open consortium and is currently helping to define the X/Open Distributed Transaction Processing standard to maintain our leadership position in OLTP.

HP is committed to providing key innovative standards such as OSF/MOTIF and OSF DCE on the HP 3000. The operating system offering of OSF is called OSF/1. Because OSF/1 is POSIX compliant, HP 3000 will be able to run applications based on OSF standards.

HP also has a strong commitment to de facto standards. This is needed because even X/Open does not cover several of the popular products on the market today such as Novell LANs, SNA, third party databases and 4GLs today.

The final force driving HP to open systems is its commitment to coexist with IBM. IBM has defined its proprietary standards architecture called the Systems Application Architecture that consists of Common Communication Services (CCS), Common User Access (CUA), and Common Programming Interfaces (CPI) specifications. Our focus is on compatibility with their Common Communication Services for networking interoperability.

APPLICATION PORTABILITY

Let's now look at the second key requirement for Open Systems -- Application Portability. Application portability is the ability to provide hardware and software vendor independence for increased solutions flexibility and reduced software development costs.

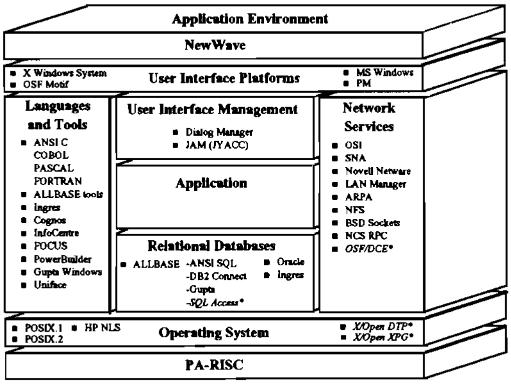
Application portability answers several questions that are important to MIS Directors when evaluating product technologies available on vendor platforms. Are applications portable from different platforms to the HP 3000 via standards? Can new applications be developed that are portable across different platforms? And even more important, can new applications be developed that are portable and state-of-the-art taking advantage of the latest technologies in graphical user interface support, client/server computing with relational databases, and distributed computing?

As you will see, it's important for a vendor to be able to provide a software development framework of different standards-based products and services to meet these needs.

The HP 3000 opens systems environment framework recognizes the need to provide networking and Application Programming Interfaces (APIs) to facilitate interoperability and portability of applications in the heterogeneous computing environments.

This framework is targeted for use primarily by application developers. It is important for products and standards to be defined for each of the components that may interface to the application. This includes APIs for the operating system, the user interface, the network services, the databases and the languages and tools. Notice that although operating system standards are important, they are a small part of the whole framework.

HP 3000 Open OLTP Environment



* *Planned for 1992*

HEWLETT PACKARD

The above figure summarizes the HP 3000 Open Systems Environment which consists of standards-based products and services in each of the major areas. The HP 3000 supports (or will support shortly), all the key open systems application interfaces and services. These standard interfaces facilitate a high degree of modularity or "plug and play" for application portability. As a result, the HP 3000 provides a well-rounded set of open system solutions. This environment provides the following features and benefits:

1) Support for industry and de facto standards based products for new application development.

   All the products and services listed are supported on multiple vendor platforms or are implemented with a standards-based interface. This allows new applications to be portable between other open systems environments. In particular, applications developed using these products and services are portable between HP 3000 and HP 9000 systems.

2) Support for 3rd Party Databases and 4GL Portability

   Examples are HP's open SQL solutions with Allbase and third party databases and tools. It is important to note that database and toolset selection are the two most important considerations for portability. In addition, multivendor 4GL tools allow for ease of application development.

3) Support for standards based graphical user interfaces

   Graphical User Interfaces are important for consistent "look and feel" in a multivendor environment, ease of use and reduced training costs. Examples are HP 3000 support for the X Window System, MS Windows, and Presentation Manager.

4) Support for state of the art client/server applications

Examples are products and services that allow for partitioning of the application across PCs or workstations and back end servers. Gupta SQL Windows and Powersoft Powerbuilder allow Windows 3.0 applications to run on a PC and have remote database access to HP 3000 servers on a LAN. Client/server Allbase application support is provided by the Gupta client/server API, a defacto standard. The HP implementation of this standard supports Netware and TCP/IP LANs.

5) Support for UNIX applications portability

Products and technologies in this framework provide support for porting UNIX applications to the HP 3000. As an example, portability of applications from an HP-UX environment to an HP 3000 environment is greatly enhanced through the use of a common C compiler. POSIX provides application portability interfaces at the operating system level. (3GL) UNIX portability is also supported with Oracle databases at the 4GL level through Ingres. These standards enable HP 3000 customers to gain access to leading UNIX applications while preserving their current investments.

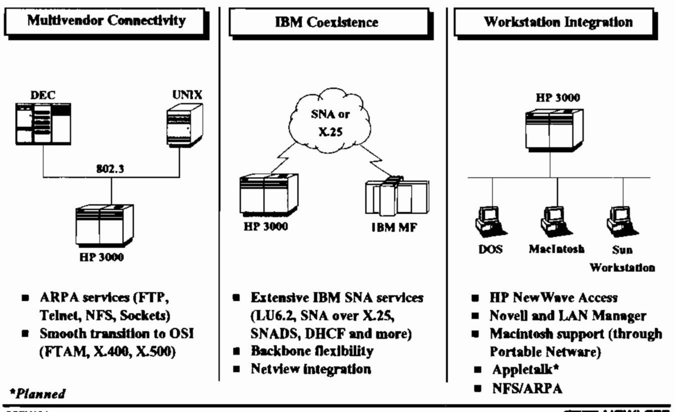
6) Support for transparent access to distributed data and services in a multivendor environment

An example of this support would be SQL Access to remote databases that allow new applications to query multiple databases on remote systems.
A variety of networking interfaces also facilitate this transparent access. An example is the Network Computing System (NCS) remote procedure call facility that has been adopted by OSF DCE for cooperative application processing. NCS is supported today on the HP 3000.

MULTIVENDOR INTEROPERABILITY

The third characteristic of an Open System is Multivendor Interoperability which is the ability to provide system networking integration with other systems including IBM, UNIX, PCs and workstations and transparent access to information across the enterprise. Connectivity and coexistence with installed multivendor systems is key.

Industry Standard Networking



- ARPA services (FTP, Telnet, NFS, Sockets)
- Smooth transition to OSI (FTAM, X.400, X.500)

- Extensive IBM SNA services (LU6.2, SNA over X.25, SNADS, DHCF and more)
- Backbone flexibility
- Netview integration

- HP NewWave Access
- Novell and LAN Manager
- Macintosh support (through Portable Netware)
- Appletalk*
- NFS/ARPA

*Planned

OPEN124

HEWLETT PACKARD

HP's NewWave computing strategy is a strong long term commitment to multivendor connectivity, IBM coexistence and PC integration. The HP 3000 supports the networking services needed to ensure integration into a variety of installed base multivendor environments.

ARPA and OSI services are necessary for UNIX and DEC connectivity. The HP 3000 supports file transfer services and virtual terminal access via these services today.

The HP 3000 has a leadership position in IBM connectivity and has continued to strengthen coexistence with IBM via a variety of SNA products, such as SNA Distributed Services (SNADS) for electronic mail exchange with IBM.

HP is considered #1 in PC integration due to it's NewWave Office product offerings and continues it's leadership position by providing integration support with the most popular LAN environments, Novell Netware and LAN Manager.

HP 3000 OPEN OLTP SUMMARY

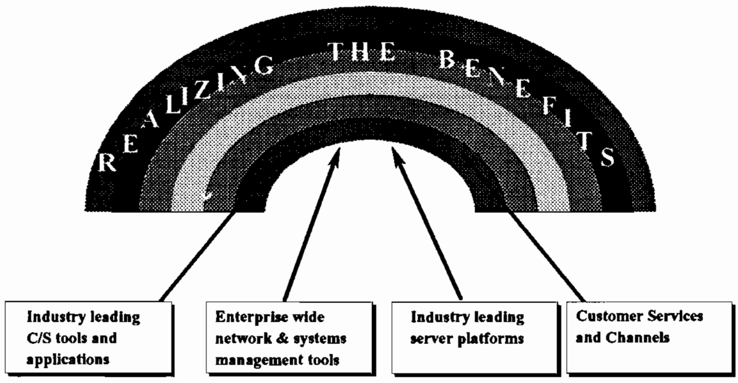
The HP 3000 strategy is to support relevant commercial open system standards to provide application portability and multivendor interoperability while providing industry-leading OLTP performance and functionality and protection of customers' investments. The strategy encompasses support for de facto market standards, consortia-based standards (X/Open, OSF), and IBM SAA interoperability standards and provides key application programming interface standards for all components of a software computing environment. In addition, the HP 3000 supports a long-term commitment to OSI, industry-wide database interface definitions (SQL Access Group), standard graphical user interfaces (for example, PM, OSF/Motif), industry-leading object management (NewWave Computing), and standard network management protocols (HP OpenView). Commitments to both standards and industry-leading OLTP will make the HP 3000 one of the premier open platforms of the 1990s.

II. DEVELOPING CLIENT/SERVER APPLICATIONS ON THE HP 3000

We have now discussed open systems on the HP 3000 and in this section will discuss client/server computing and how the HP 3000 provides an environment conducive to developing applications which take advantage of the benefits of this technology.

Let's begin with a definition of client/server computing. We find that although this technology is one of the most talked about in the industry today, it is often not well understood. Client/server is a model of computing that divides functions between client and server machines. Its allows users to maintain desktop computing while strengthening their capabilities through transparent access to files and compute power from a variety of other computers in a distributed networked system. Client/server computing is any variety of methods to distribute the user interface, the data and the application software. This technology allows the user's applications to take advantage of the combined processing power of not only the desktop computer, but all of the computers in a network.

Client/server computing combines the strengths of individual desktop computing with the established capabilities of larger centralized systems. Among the many reasons encouraging client/server implementations is a desire to use the dormant power residing on most desktops. The continuous improvement in personal computer processing power, along with the sheer numbers of PCs and workstations has resulted in tremendous processing power on the desktop which is quite often idle. In the past, application software was not written to use the embedded processor power of your desktop device. Personal computers, for example, were used as "brainless" terminals. Now, with client/server computing, applications use the desktop computer's processor to perform some of the application processing and also to process the user interface. This is how client/server computing begins to provide some real benefits. You can continue to use your existing or future personal computer for single-user applications and use its full power to improve the processing of applications used by many individuals.


REALIZING THE BENEFITS
Industry leading C/S tools and applications
Enterprise wide network & systems management tools
Industry leading server platforms
Customer Services and Channels

Commercial Systems Division
CSMDL50 CF;RO 02/92

HEWLETT PACKARD

Many customers are moving towards client/server computing due to the increased productivity that it offers end users through easy to use graphical user interfaces and the investment protection provided by standards based components such as SQL, Netware, LAN Manager and Microsoft Windows. Implementing client/server technology requires a combination of tools, applications, network and systems management and a vendors support to truly realize the benefits of client/server computing. With all of these features built on an industry leading open platform like the HP 3000, HP is in a position to really partner with its customers in their transition to client/server computing.

HP's strategy is to provide the components, tools and solutions that make it possible for customers to evolve over time from their existing computer environment, or make a direct move, to a client/server computing environment. HP has been aggressively pursuing open client/server computing because of the increased productivity and investment protection that it brings our customers. Initially, our product offerings focused on decision support/information access applications, such as NewWave Office. This offering had sort of a "look and feel" of client/server applications, but did not update data on the HP 3000.

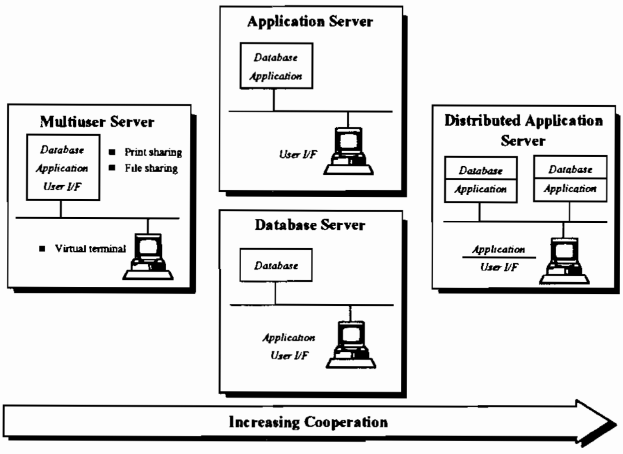
Our next step was to provide tools that protect customers investments in existing applications while moving them forward in an evolutionary manner to open client/server computing by taking existing OLTP applications and applying a graphical user interface. We also introduced tools that allow customers to create new applications that run on a client and update data on an HP 3000 server.
We will discuss these tools later in the paper.

The future of open client/server computing lies in defining and implementing standards that allow customers to integrate all of the diverse computing resources throughout an enterprise. At present, many applications are still "islands of automation" since they cannot be integrated in with other applications that were developed using different databases, networking, or operating systems. HP is driving standards for SQL database interoperability and portability through our participation in the SQL Access Group and standards for Distributed Transaction Processing (DTP) through our involvement in the X/OPEN DTP group. In the future, HP will provide products that support each of these emerging standards.

In reality, client/server computing is really not where the industry predicted it would be by 1992. Originally, leading industry consultants predicted that the client/server technology would take off by 1992, but now predictions indicate that it will be 1994-1995 before the bulk of the industry is implementing client/server.

Client/server computing is independent from the technologies used to implement it. You don't have to wait for the perfect technology to implement. Instead, HP encourages its customers to begin today to put the plans and road maps in place to move towards client/server computing. Our goal is to ease customers to client/server computing in a way such that investments are protected for use in tomorrow's environments.

Server Configurations



Commercial Systems Division
CSMDL08 CF;RO;GD 01/92

HEWLETT PACKARD

CLIENT/SERVER MODELS

Let's look at the different ways that we could develop client/server applications. The simplest is the Multiuser Server which has been around for several years. This is primarily virtual terminal access to a server over a LAN for tasks like file or print sharing. This is accomplished by using networks such as LAN Manager or Novell Netware. With this type of server, the client MIPs are not taken advantage of at all as the database, application and user interface logic all reside on the server.

The next two servers, the Application Server and the Database Server really evolved at about the same time depending on whether the users were coming from a mini-centric environment (Application Server) or a PC-centric environment (Database Server). Both of these models really begin to off load the server and thus take advantage and make good use of the MIPs and actual processing power of the clients.

We see the Application Server as a step to move existing applications forward by putting a new user interface on the application. The application and database remain on the server. Examples of this type of server involve VPLUS Windows or use of a Windows 3.0 terminal emulator.

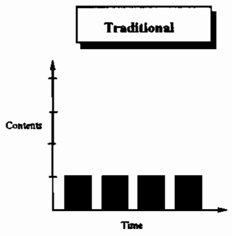
The Database Server implementation involves moving the application and user interface logic to the client with only the database on the server. We will discuss popular 4GLs available today for implementation of this model.

Over the next few years, we see the industry moving towards the Distributed Application Server model. This model involves true cooperative processing as the application (and data) is split across potentially multiple servers and clients. As a result, this server provides the best possible performance and allows you to integrate the largest number of data sources within an organization.

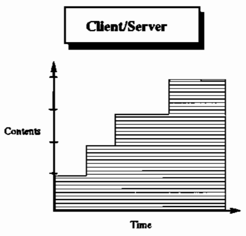
We will now discuss the Application Server, the Database Server, and the Distributed Application models as well as the tools available and the alternatives for application development within each.

Increased End User Productivity

Graphical User Interface and Windows



- **One context at a time**
- **Not intuitive**
- **Less productive**



- **Reduced context switching**
- **Quicker access to information**
- **Reduced training time**
- **More tasks accomplished**

Commercial Systems Division
CSMDL11 CF;RO;GD 01/92

HEWLETT PACKARD

APPLICATION SERVER MODEL

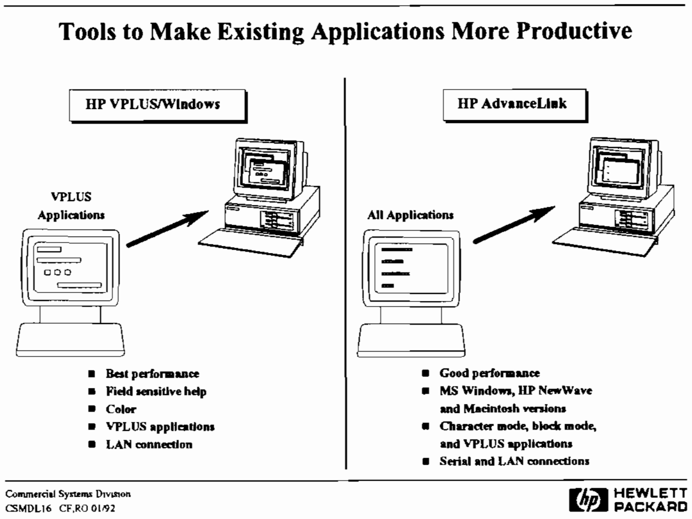
With the Application Server Model, the user access is distributed out to the client. With this model, the user will see <u>an increase in end user productivity</u>.

Two advantages of this model are windowing and graphical user interfaces. In the traditional host-terminal model, the user would run an order entry screen, an order status application, a customer look-up application, etc. and would be forced to run each program sequentially. With Windows, the user can have multiple applications up and running at the same time. This allows for much quicker access to data and reduced context switching. In addition, it allows the user to design applications with a common look and feel making training easier by avoiding the need for users to learn multiple interfaces.

The first tool for developing applications using the Application Server Model is MS-Windows 3.0 Developer's Kit which is a standard product available from Microsoft. We direct the reader to the Microsoft materials for further clarification of this kit.

In certain instances, rather than creating a new application, you may want to move across new application modules like a decision support module to a client/server environment. One of the tools that HP has available today is HP NewWave Access where, using point/click, and NewWave features such as hot link and agent you can automatically extract data from multiple databases and combine them into compound documents.

The HP 3000 has adopted the HP OSF/Motif style guide and has added support for the industry standard X Window System. X Windows allows HP 3000 users to develop Motif applications that front-end the HP 3000 resulting in the same "look and feel" whether the application is running on the HP 3000 or a UNIX system.



Tools to Make Existing Applications More Productive

HP VPLUS/Windows

VPLUS Applications

- Best performance
- Field sensitive help
- Color
- VPLUS applications
- LAN connection

HP AdvanceLink

All Applications

- Good performance
- MS Windows, HP NewWave and Macintosh versions
- Character mode, block mode, and VPLUS applications
- Serial and LAN connections

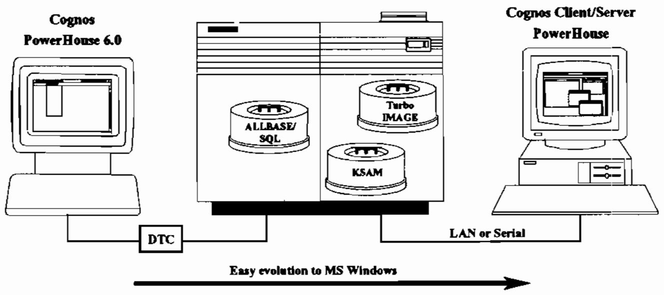
Commercial Systems Division
CSMDL16 CF,RO 01/92

HEWLETT PACKARD

HP has an objective to protect our customer's investments in existing applications while moving them forwards in an evolutionary manner to open client/server computing. HP VPLUS/Windows and HP Advance Link for NewWave/Windows and the Macintosh meet this objective. Both are windows based terminal emulators and help provide a means for making existing applications more productive. VPLUS/Windows provides the highest performance for customers who want to integrate their existing VPLUS applications into a MS-Windows environment. Applications can also be enhanced to provide field sensitive help and color.

If customers have a large number of PCs already installed, they can effortlessly integrate their terminal based applications into a NewWave, Windows, or Macintosh environment using Advance Link for Windows. Advance Link provides access to multiple HP 3000 applications at the same time if used over a LAN or provides single application access if used with a serial port. Advance Link supports character mode, block mode, and VPLUS applications. NewWave support provides a highly productive "drag and drop" capability for script file execution.

Client/Server COGNOS PowerHouse



Common CASE tool for TurboIMAGE, KSAM and ALLBASE/SQL

Commercial Systems Division
CSMDL17 CF;RO 01/92

HEWLETT PACKARD

There are some additional tools that allow you to enhance a user interface for a windows environment. One is the Cognos PowerHouse 7.0 tool which allows you to take a PowerHouse 6.0 application and add a full Windows 3.0 user interface. PowerHouse is the leading 4GL on the HP 3000 and the DEC VAX. COGNOS recently announced PowerHouse support for the AS/400.

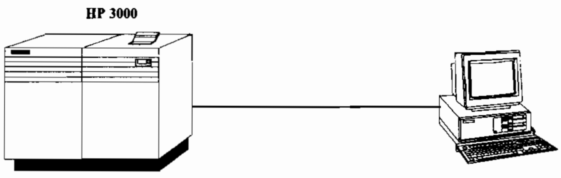
Cognos' PowerHcuse 6.0 tool allows you to develop simple, field mode applications with pull-down menus. It provides direct access to all HP files (Allbase/SQL, TurbolMAGE, KSAM and MPE). This allows for easy substitution of ALLBASE/SQL in the future. Read/write access to DB2 data is also supported through the HP ALLBASE/DB2 Connect product.

The client/server version of PowerHouse (7.0) provides an evolutionary path for current PowerHouse applications by enabling developers to upgrade to a client/server environment without reprogramming existing applications. It enables the creation of new applications while also allowing developers to easily move PowerHouse 6.0 applications to a Windows 3.0 environment, much like VPLUS/Windows does for VPLUS applications. In addition, the client/server PowerHouse release will be a complete toolset for developing applications, forms and reports. It will also support access to graphical data.

APPLICATION SERVER MODEL: ADVANTAGES AND DISADVANTAGES

The advantages that we see for using the Application Server Model for existing applications include a very easy migration to a graphical user interface through the use of VPLUS/Windows or Advance Link/Windows. End users gain immediate productivity increase through the graphical user interface which makes it very cost effective. In addition, a customer preserves his investment in existing applications. Users also see hardware and software investment protection through products like Cognos PowerHouse which allow users to preserve their investments. A disadvantage to this model is that it is really not optimized for client/server computing. In addition, application rewrites are necessary for some of the tools.

Database Server Model
Distributed Data



- **Data storage and access**

- **User interface display and management**
- **Application logic**

← **Communications** →

Commercial Systems Division
CSMDL21 CF,RO 01/92

HEWLETT PACKARD

DATABASE SERVER MODEL

The next model, the DataBase Server model, grows out of environments with a large number of PCs already in place previously used in an environment like the Multiuser server (file or print sharing). In this model, the full application runs on the client while only the database runs on the server. Many tools are available on the market today to develop applications using this model.

When implementing applications using this model, the developer must take into account programmer productivity, long term standards support, range of clients supported, range of databases supported, and investment protection of existing hardware and software. For this model, we will discuss 4GL tools which increase programmer productivity by hiding development complexities of APIs, networking, or database access. If the programmer uses these tools, he may not need to know C or have SQL programming knowledge.

It is important to look for standards where they are available today. Sometimes a developer may see a tradeoff between gaining productivity through the use of 4GLs and application performance productivity by using a combination of 3GLs and coding to standard APIs where they exist.

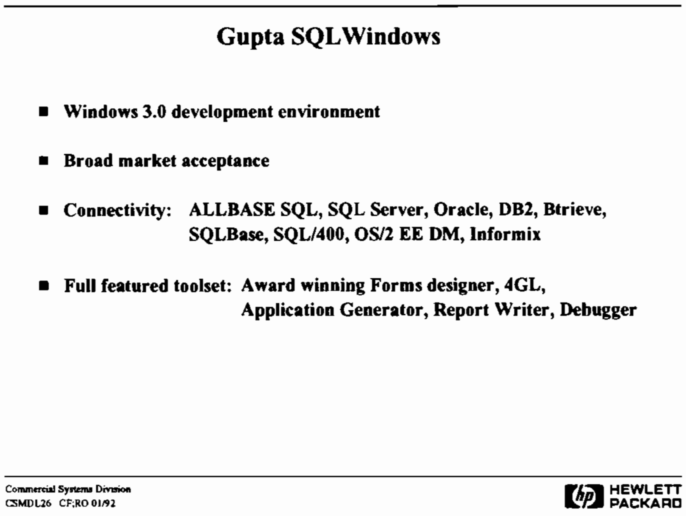
When developing applications using the Database Server Model, you can choose to "build your own applications" by using 3GLs and standard APIs. Examples include the creation of a user interface with NewWave, Windows 3.0, and Presentation Manager. After development of the user interface, code is generated by the developer to write directly to the network APIs like NetWare SPX/IPX, LAN Manager Named Pipes, and Berkeley Sockets to access the database across the network on the server.

If the developer does not want to write to the network APIs, a higher level API can be selected such as the Allbase/SQL PC API or Microsoft ODBC (which was recently chosen by the SQL Access group as a standard SQL API for C/S development).

The Allbase/SQL PC API is a collection of Dynamic Link Libraries (DLLs) that reside on a PC and contain the necessary Application Programming Interfaces (APIs) to support two 4GLs (Gupta's SQL Windows and Powersoft's Powerbuilder) and the PeopleSoft human resources application with ALLBASE/SQL.

The HP ALLBASE/SQL PC API is based on Gupta's client/server SQLBase API.
Through a series of SQL calls passed over the network by using Allbase NET, this API gives you read/write access to Allbase and read access to TurboIMAGE using HP Allbase/TurboCONNECT and will allow you read/write access to DB2 using DB2 Connect. Gupta themselves sell routers to SQL Server, Oracle, DB2, Btrieve, SQLBase, SQL/400, OS/2 EE DM, and Informix.

Another alternative for developing new applications for client/server by using the Database Server Model is the use of sever tools already available on the HP 3000. These tools are industry leading 4GLs and greatly simplify application development over using 3GLs or coding using APIs. These tools include Gupta SQL Windows, Powersoft PowerBuilder, Ingres/Windows 4GL and our most recent announcement, Uniface.



Gupta SQLWindows

- **Windows 3.0 development environment**
- **Broad market acceptance**
- **Connectivity: ALLBASE SQL, SQL Server, Oracle, DB2, Btrieve, SQLBase, SQL/400, OS/2 EE DM, Informix**
- **Full featured toolset: Award winning Forms designer, 4GL, Application Generator, Report Writer, Debugger**

Commercial Systems Division
CSMDL26 CF;RO 01/92

HEWLETT PACKARD

Let's take a look at the 4GL tools that are available. Gupta Technologies with their SQL Windows is recognized as one of the leaders in the field for client/server applications development. They have broad market recognition and acceptance.

The SQL Windows toolset from Gupta provides a complete development environment to create Microsoft Windows applications that update an ALLBASE/SQL database. SQL Windows includes a forms generator, 4GL, report writer and debugger. It is a flexible toolset and provides support for entry level to advanced programmers.

At a PC level, SQL Windows can also update a local PC database, Gupta's single user SQLBase, in conjunction with accessing a database server.

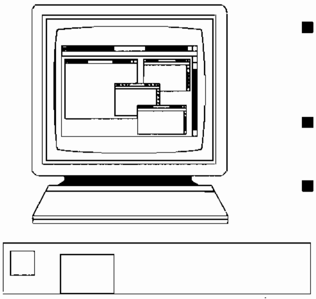
Powersoft PowerBuilder

- **Windows 3.0 development environment**
- **Highly graphical user interface**
- **Databases supported: ALLBASE SQL, Oracle, Microsoft SQL Server, SQLBase, Sybase SQL Server**
- **Flexible development environment: Screen painter with fonts and colors, 4GL, scripts, debugger**

Another 4GL tool available today is the Powersoft PowerBuilder 4GL. PowerBuilder is a highly graphical development environment that allows developers to visually "paint" applications without any programming. SQL data can be selected using visual representations of the tables without writing any SQL statements. The PowerBuilder toolset includes the "painters" for screen generation, a 4GL, and a debugger.

Support for PowerBuilder with ALLBASE/SQL is provided by the new HP ALLBASE/SQL PC API product. In addition to updating information in ALLBASE, PowerBuilder can work with HP TurboCONNECT to give read access to TurboIMAGE data. Read/write access to IBM's DB2 database using HP ALLBASE/DB2 Connect is also possible. Along with ALLBASE/SQL PowerBuilder can create applications that update Oracle Server, Gupta's SQLBase, Microsoft's SQL Server, and Sybase's SQL Server.

INGRES/Windows 4GL



- **"Programmer-friendly" windowing technology**
- **UNIX workstation clients**
- **Client-server applications with HP ALLBASE/SQL**

Commercial Systems Division
CSMDL31 CF:RO 01/92

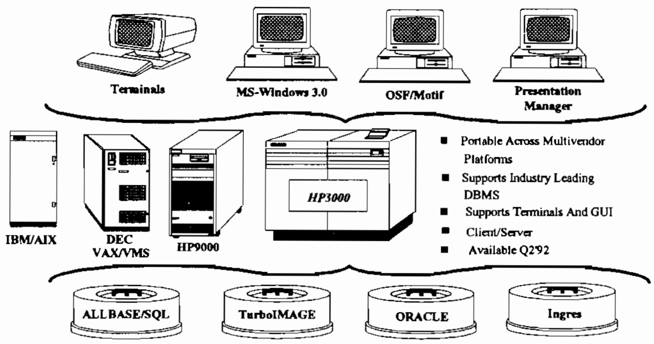
HEWLETT PACKARD

INGRES/Windows 4GL has revolutionized client/server applications development for customers who have standardized on UNIX workstation clients to access servers. The product accomplishes this by integrating a visual user interface editor with an object oriented 4GL. The visual editor lets developers build graphical user interfaces quickly, while the 4GL reduces coding.

All application elements are catalogued in the INGRES data dictionary with complete documentation for accurate design and maintenance. Application elements can be shared easily between applications, leveraging development activities.

Connectivity to ALLBASE/SQL is provided through the INGRES Gateway product. At present, INGRES/Windows 4GL should be used for decision support applications. Over the next year, the INGRES Gateway will be tuned to provide support for transaction processing applications with ALLBASE/SQL.

Uniface



- Portable Across Multivendor Platforms
- Supports Industry Leading DBMS
- Supports Terminals And GUI
- Client/Server
- Available Q2'92

Ideal For Major Accounts And VAB

Commercial Systems Division
CSMDL30 CF;RO 01/92

HEWLETT PACKARD

A new tool which we have recently added support for on the HP 3000 is Uniface. Uniface spans support across a broad range of clients (terminals, MS-Windows 3.0, OSF/Motif, and Presentation Manager). In addition, it is portable across multivendor platforms including IBM/AIX, DEC VAX/VMS, the HP 9000 and the HP 3000. Industry leading DBMSs are also supported.

The Uniface tool is quite powerful in that it allows you to run across multiple clients while providing support for multiple databases. When developing graphical user interfaces, the Uniface tool adapts itself to the client on which you are running.

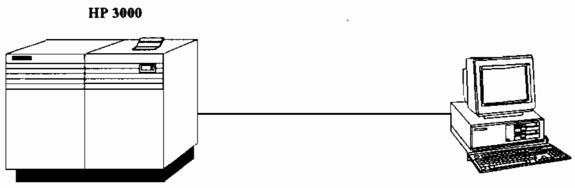
DATABASE SERVER MODEL: ADVANTAGES & DISADVANTAGES

In summary, the Database Server Model is advantageous in that tools are available on the market today which greatly simplify the developer's task. These tools run across a variety of databases and a variety of clients making them highly portable.

In addition to the 4GLs for developing applications using the Data Server Model, there are also some capabilities for incorporating the distributed applications technology. An example of this capability is stored procedures.

When using 4GLs, developers must keep an eye on performance as it could deteriorate as SQL statements are continually shipped back and forth across the network. Another disadvantage comes about if no 4GL tools are used as the resulting code generation task becomes quite complex.

Distributed Application Server Model



- **Data storage and access**
- **Application function**

- **User interface display and management**
- **Application function**

Communications

Commercial Systems Division
CSMDL38 CF:RO 01/92

HEWLETT PACKARD

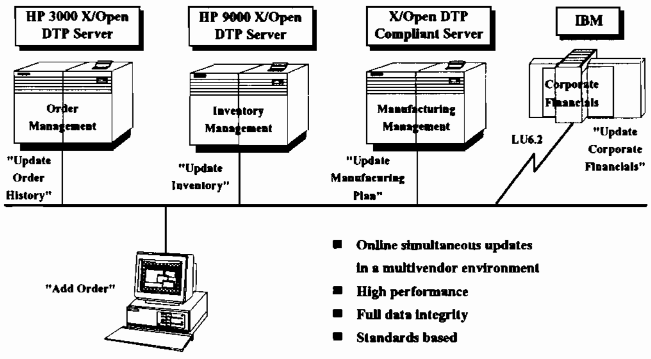
DISTRIBUTED APPLICATION SERVER MODEL

The final model and the one which we see the industry moving towards over the next couple of years is the Distributed Application Server model in which the application is split between the client and the server or potentially amongst multiple servers. In addition, the database may be split amongst multiple servers. This model takes advantage of the availability of low-cost computing power in the form of intelligent workstations (PCs, UNIX workstations, etc.) to offload a part of the application processing much like the Database Server Model, but actually goes a step further in that the application and data are spread throughout the enterprise. With this model, we see a true optimization of computing resources as information flows from wherever it is stored in the enterprise to wherever it is needed.

While we look forward to wider availability of enabling technologies for this model (such as OSF/DCE), there are some technologies that can be used today to build applications for this model. One of these technologies is message passing (Interprocess Communication) where the developer opens a communications pipe and creates messages to be shipped back and forth between clients and servers. Examples of this type of message passing could be done by using LAN operating systems such as LAN Manager or Netware or by using Berkeley Sockets.

One of the emerging paradigms for this model is to use Remote Procedure Calls (RPCs) in place of IPCs. RPCs are favored by developers because of their intuitive nature. Developers are more comfortable with making procedure calls than creating messages and using pipes. NCS/RPC was accepted as a part of the OSF/DCE offering. Another RPC which is also widely accepted in the industry is Netwise which is also available on the HP 3000.

X/Open DTP: Enterprise Wide OLTP Applications



- Online simultaneous updates in a multivendor environment
- High performance
- Full data integrity
- Standards based

Transarc technology and HP NCS provide the base

Commercial Systems Division
CSMDL45 CF,RO 01/92

HEWLETT PACKARD

Along with OSF DCE, we see a need to distribute transactions across an enterprise. As companies begin to compete more on quick time to action in responding to customer's needs, it becomes critical that the information systems throughout a company can access and update data on a real time basis. Today many databases are updated at night by batch jobs because an application is unable to update data on-line that resides on a different machine or in a different database from that in which the application uses.

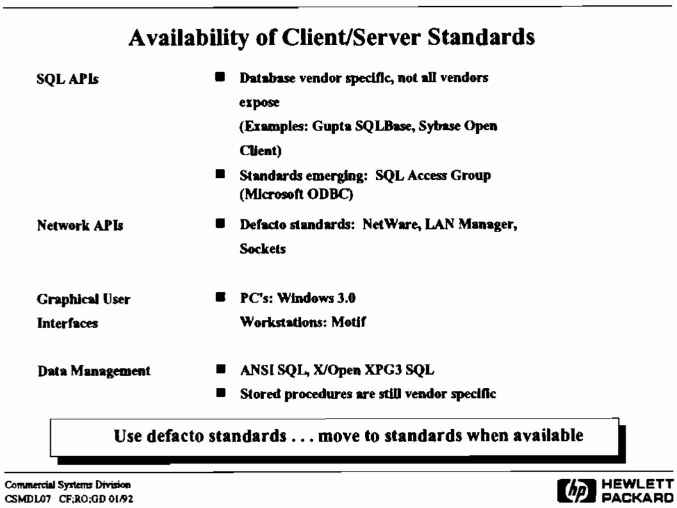
What is needed is a set of standards that defines how an application coordinates transactions and updates data across multiple vendors databases, operating systems, and data communications protocols. X/Open is addressing this issue by formulating standards for Application Programming Interfaces (APIs) for Distributed Transaction Processing (DTP) applications. HP will be providing products for both MPE/iX and HP-UX that adhere to the X/Open DTP standards soon. To achieve this, HP is licensing technology from Transarc Corporation. The technology from Transarc is built on top of HP's NCS Remote Procedure Call (RPC) and is consistent with our strategy to use the OSF DCE as a framework for distributed applications. The distributed transaction management technology from Transarc will be integrated in with the existing transaction management facilities in MPE/iX and extend them to a multi-system, multi-vendor environment.

Other benefits that we see for DTP are a high level of performance. If we look at how customers are competing today on reduced cycle time (quicker time to action) and increased customer service, having real time access to this information is a big benefit.

DISTRIBUTED APPLICATION SERVER MODEL: ADVANTAGES AND DISADVANTAGES

What are the advantages to distributing the application? An obvious advantage is that performance is optimized beyond the capacity offered by the other models discussed. As a developer strives to protect his investments, an encouraging note is that standards like OSF/DCE are emerging which also is an advantage. The use of these standards will form the basis to create enterprise wide applications. In addition, it is through the use of this model that a user will realize the full benefits of distributed computing through access to all of the resources throughout an enterprise.

Currently, the disadvantage to this model is that not many tools have incorporated NCS or other technologies leaving developers with 3GLs as an option for development. Because of the lack of tools in this area, we consider this the most complicated model to develop.



Availability of Client/Server Standards

SQL APIs	■ **Database vendor specific, not all vendors expose (Examples: Gupta SQLBase, Sybase Open Client)** ■ **Standards emerging: SQL Access Group (Microsoft ODBC)**
Network APIs	■ **Defacto standards: NetWare, LAN Manager, Sockets**
Graphical User Interfaces	■ **PC's: Windows 3.0 Workstations: Motif**
Data Management	■ **ANSI SQL, X/Open XPG3 SQL** ■ **Stored procedures are still vendor specific**

Use defacto standards . . . move to standards when available

Commercial Systems Division
CSMDL07 CF;RO;GD 01/92

HEWLETT PACKARD

AVAILABILITY OF CLIENT/SERVER STANDARDS

As we have already mentioned, HP's strategy is to truly partner with their customers in the move towards client/server computing while maintaining and protecting the customer's investments. Standards provide one of the best methods to protect investments. In general, the entire client/server arena has undergone little standardization. The above figure shows the various standards (including defacto standards) that can be used in a client/server environment.

The database is an important area to provide standards. SQL has emerged as the leading standard in this area. Within SQL itself, there really aren't any standards. There are however, standards emerging for APIs to make SQL calls from a client to an SQL database residing on a server. HP has already released the Allbase/SQL PC API for the HP 3000 (discussed earlier) and plans to add the Microsoft ODBC to the offering in the near future. The Microsoft API has recently been chosen by the SQL Access group.

In the networking area, few standards have been established. There are defacto standards like Novell's Netware, LAN Manager, and Berkeley sockets which are widely used in the industry. In the same manner, graphical user interfaces like Windows 3.0 for PCs and MOTIF for workstations are becoming popular.

In the data management area, there are SQL standards like ANSI SQL, X/Open, XPG3 SQL, but a popular technology, stored procedures, is still in the proprietary stage. With stored procedures, the developer can actually store multiple SQL statements within the database thus enhancing performance and allowing a higher level of data integrity. Performance is enhanced because the logic for transactions can be stored along with the actual data for that transaction. For example, if you are doing an "add order" and all transaction logic (update order history table, update customer file) lives close to the data, all work happens right at the server level rather than shipping the add order statements back and forth across the network. The level of data integrity is increased with stored procedures because all rules (such as type validation) can be stored within the database. In the past, developers were forced to perform those types of verifications within an application program itself. In addition, as "rules" changes (i.e., if new types are added) the change is made once to the "rule" within the stored procedure.

HP's philosophy with regards to standards is to support key standards as they become available while supporting defacto standards in the interim.

III. CLIENT/SERVER VAB APPLICATIONS FOR THE HP 3000

In addition to providing tools for developing client/server applications using the HP 3000, HP is continually recruiting state of the art client/server VAB applications for the HP 3000. Solutions are available today in accounting, human resources, manufacturing, and distribution. The following are some examples of premier solutions available on the HP 3000 which make use of the client/server technology.

DATALOGIX

For the manufacturing environment, Datalogix CIMPRO/R is an advanced world class PRS (Process Requirements Planning) software that has been built on the same proven functionality of CIMPRO. CIMPRO/R provides a fully integrated business and manufacturing package that can be used in both distributed or consolidated environments.

CIMPRO/R includes modules that support formula/recipe management, regulatory compliance, computer-aided formulation (CAF), laboratory business support, integrated quality control, costing, activity-based accounting, inventory control/management, order processing, capacity requirements planning, material requirements planning and process operations control.

The product has been designed to provide an ergonomic and international user interface design through GUI, field level help, function key support, mouse support, Natural Language Interface and window support.

FOURTH SHIFT

FOURTH SHIFT is a leader in client/server manufacturing software with over twenty manufacturing and financial application modules covering all areas of business. Modules include inventory control, order management, MRP II, costing, order entry, accounting, and purchasing. FOURTH SHIFT, designed for decentralized manufacturers, increases productivity by making the systems more responsive to the needs of local users. Application robustness is not a concern, market researcher Datapro claims "FOURTH SHIFT's power rivals most systems (mini and mainframe) that we have seen." Discrete, repetitive, and process manufacturers, including more than 20% of the Fortune 500 have FOURTH SHIFT installed today.

COLLIER-JACKSON

Collier-Jackson, a leader in accounting and human resources software, has re-engineered their software to take full advantage of new client/server technologies. The client in the Collier-Jackson client/server environment is an IBM compatible PC that will run the on-line portion of the application. The server is a host system running MPE/iX that manages the database and runs batch applications. Collier-Jackson has also taken steps to protect their customers' current software investments. The new client/server versions have been engineered to run in both the new client/server mode as well as the traditional host based mode. Customers looking for an easy migration to client/server can take advantage of the extra effort Collier-Jackson has taken to allow users to utilize both modes at the same time. You can run an on-line Collier-Jackson application operating on a PC while another user is running the identical application on the host from a terminal. Both users are processing information from the central host managed database. This will allow customers to take advantage of the power of the PCs on the desktop at their own pace, without the need to entirely replace current software.

These new applications have been designed as open solutions. They take full advantage of graphical user interfaces (JAM) and industry leading SQL databases including ALLBASE/SQL.

ORACLE FINANCIALS

Oracle has designed and tested a true client/server version of financials specifically for the HP 3000. Oracle financials is a complete, tightly integrated family of cross-industry accounting and management applications. Modules include GL, AP, AR, purchasing, inventory, assets, and Oracle personnel. Because it is built on the Oracle DBMS using SQL*Forms, Oracle financials is extremely easy to use and customize. Oracle financials provides easy-to-use graphical Lotus and Macintosh style menus, on-line help, and pop up windows. Oracle has quickly established itself in the market with hundreds of installs spread throughout the world.

MITCHELL HUMPHREY

Mitchell Humphrey is a leading supplier of financial accounting solutions on the HP 3000. Mitchell Humphrey targets large companies who can take advantage of their rich functionality. They have recently upgraded their solution to take advantage of the latest graphical user interface and relational database technology. Mitchell Humphrey offers a full suite of financial applications including GL, AP, AR, fixed assets, purchasing, inventory, and government compliance.

PEOPLESOFT

PeopleSoft was first introduced during our December, 1990 product introduction. The PeopleSoft solution fully addresses human resources and payroll department needs for the 1990s. PeopleSoft HRMS is fully integrated. Each functional component -- Payroll, HR, and Benefits -- shares the same database, user interface, reporting tools, and customization facilities. The PeopleSoft product was developed from the beginning to be a true client/server application. The graphical user interface gives the system a comfortable look and feel while the table driven design makes customization very simple. PeopleSoft's client/server design and graphical customization tools enable implementation in half the time and at half the cost of competitors. The PeopleSoft application is being recognized throughout the industry as being clearly a step ahead today of any of its competitors because of its standards based, client/server design.

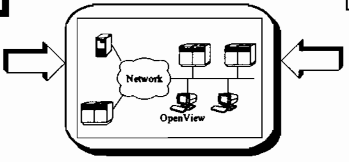
DISTRIBUTION RESOURCES COMPANY

Distribution Resources Company announced in 1990 the availability of it's Customer Service Workstation (CSW) application for the HP 3000. This application is a true client/server application utilizing an HP 3000 server along with an MS-DOS client platform. The CSW provides a highly interactive order entry system with accounts receivable and inventory management functions. Each workstation has it's own database of static information that supplies 95% of all information needed to process local data and run programs within a windowing environment. The other 5% of dynamic information is available on an HP 3000 database server. This product is targeted at wholesale distributors in office products. paper, industrial, electrical, building supplies, and medical industries.

Managing Enterprise Wide Client/Server Environments

Systems Management

- Remote Server Management
- PC Backup Management



HP OpenView

Network Management

- LAN Management
- WAN Management
- SNA Management

Commercial Systems Division
CSMDL41 CF;RO 01/92

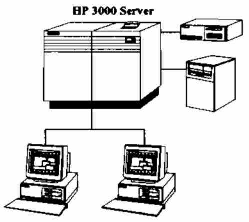
HEWLETT PACKARD

IV. MANAGING ENTERPRISE WIDE CLIENT/SERVER ENVIRONMENTS

Now that we have discussed various methods for developing applications in a client/server environment, the next step is to discuss tools and methods for managing this new and often complicated environment. While the benefits of client/server computing make its adoption attractive, the right tools are necessary to manage an organization's new environment. In this next section we will discuss HP's OpenView product which is an umbrella of wide spread tools which have been recently adopted hy OSF for its Distributed Management Environment (DME). Within the OpenView environment, HP has combined tools for systems management (remote server management and PC Backup management) and technology to manage various types of network links (LAN, WAN and SNA).

PC Management Tools

Centralized PC Backup



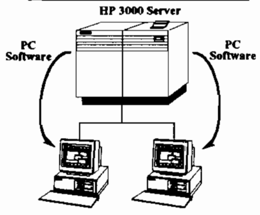
ExpressBack/3000 from Harris & Paulson

- PC Hard disk backup
- Full data protection
- HP OfficeShare or HPLAN Mgr. Networks

Plan-B from Quest Software

- PC Hard disk backup
- Full data protection
- HP OfficeShare, HPLAN Mgr., and HP Netware/XL Networks
- Backup while PC is in use

PC Software Distribution



HP Software Vendor

- Automates distribution and installation of PC software
- Maintain statistics on users
- HP Resource Sharing/V and /XL, HPLAN Mgr/X and /XL, and Netware 386

Commercial Systems Division
CSMDL42 CF,RO 01/92

HEWLETT PACKARD

With the move to client/server computing, new problems are introduced for systems management. In addition to worrying about managing the servers, MIS faces the task of providing a means for managing the clients. How does the MIS staff protect the valuable data that now resides on PCs when users, not MIS, are responsible for backing up their own PCs? ExpressBack/3000 from Harris & Paulson and Plan-B from Quest Software solve this problem. These products provide centralized backup of PC hard disks from the HP 3000 server. This allows MIS to provide full data protection for users' valuable data on the PC.

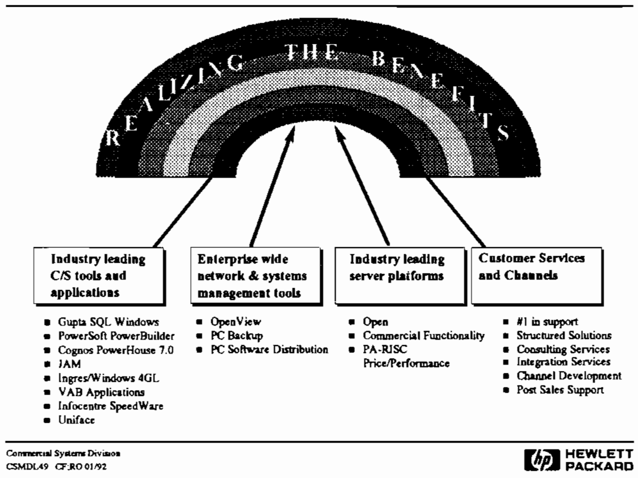
ExpressBack/3000 from Harris & Paulson provides unattended PC backup for MPE/iX and MPE V servers running HP Office Share of HP LAN Manager networks. An easy to use windows-oriented user interface makes managing backups simple. ExpressBack/3000 allows backups to be scheduled so that they do not interfere with PC users' work. Full and partial PC backups are supported on single or multiple disk drives, and in all or selected directories or subdirectories. Summary reports and audit trails of all files backed up for each PC are produced. Customized reports can also be created for specific users, groups or organizations.

With Plan-B from Quest Software PC backups are initiated and managed by the HP 3000 server even while the PC users are running applications. HP NetWare/iX, HP LAN Manager/iX and HP Office Share networks are supported. Plan-B provides turnkey backup capabilities which will selectively archive a set of PCs. Plan-B consists of both host and PC software which transparently execute and manage a high performance (3Mb/minute) backup over the network to an MPE/iX server. Flexible scheduling permits backups at any time of day and audit logs confirm the PC backup operations. Plan-B supports all HP tape drives or the rewriteable optical disk jukebox in conjunction with HP TurboStore/iX. Backups can be saved on disk providing immediate retrieval by PC users without tape intervention.

Another system management issue in moving to client/server computing is PC software management. How does MIS ensure that users have the correct version of the software that they need? Think of the wasted productivity for either MIS or end users when installation is done manually one PC at a time. HP Software Vendor solves this problem by automating the distribution and installation of PC software.

Centralized control enables system managers to maintain information on each user such as names, configuration details, packages installed, location and phone numbers. This along with central auditing of software copies vended versus copies licensed greatly simplifies and reduces the resources needed for the yearly audit of PC software.

Since installation is controlled centrally, customers have the assurance that software versions across the network are compatible. PC administrators no longer will be uncertain of what software versions they are maintaining. Users will be better satisfied with the improved support responsiveness. HP Software Vendor is supported with HP Resource Sharing/iX and HP LAN Manager/iX, HP Resource Sharing/V, NetWare 386, and HP LAN Manager/X networks.



V. REALIZING THE BENEFITS OF CLIENT/SERVER COMPUTING USING THE HP 3000

HP believes that it is in a position to form a true partnership with a customer embarking on the task of transitioning or developing a client/server computing environment.

We have already discussed in detail the industry leading tools available for developing client/server applications using the HP 3000 as a server. In addition, we have successfully recruited a wide range of applications from 3rd party VABs.

Along with the move to client/server computing, HP has realized the customer's need to manage his new environment. We will continue to add network and systems management tools and services as our customers needs change and standards emerge.

All of the solutions are built on an industry leading open server platform which tout benefits of strong commercial functionality offering and outstanding PA-RISC price/performance.

Finally, we realize that in order to be a partner throughout the client/server computing evolution to our customers, we must provide support services beginning in the area of up-front consulting and design and continuing through the integration and support of the environment.

We begin with a robust consulting service through our Open Software Environment (OSE) recently announced. The OSE services include planning and design workshops where the customer meets with our expert Open Client/Server consultants either in the factory or in the field to chart out a successful course or road map to Open Client/Server environments.

Finally, HP provides industry renowned support services continuing throughout the lifetime of your new applications.

This solid offering of platforms, tools, and services to truly partner with a customer, make HP the leading client/server vendor of the 1990s!

PAPER #3855

OBJECT-ORIENTED TECHNOLOGIES: A MANAGEMENT OVERVIEW

Orland J. Larson
Hewlett-Packard Company
19091 Pruneridge Avenue
Cupertino, California 95014

ABSTRACT

"Object orientation" (OO) is a new buzzword that promises to become the next "eureka" in informations systems development. MIS management and their software developers are under increasing pressure to improve productivity, increase software quality and reduce implementation time. Traditional software development methods are not always able to meet these increasingly heavy demands. Object orientation is getting attention as a viable alternative.

This paper reports on the growing body of knowledge about object-oriented technologies. It begins by reviewing some of the critical challenges facing today's enterprises, followed by the definitions, basic mechanisms and key concepts associated with object-oriented systems. Next, it explores various types of applications that benefit from this technnology. The potential benefits and the potential concerns will then be addressed, followed by the impact object-oriented technologies may have on data administration and systems development in the 90's.

INTRODUCTION

Each decade one or two key advances emerge and change the practice of software development. Object-oriented systems and methods are rapidly entering the mainstream of software engineering and systems development. Leading consultants are heralding object-oriented approaches as one of the most important trends to affect businesses in the 90's.

Software is currently lagging behind hardware capabilities and the lag is increasing. There is general agreement that conventional software tools and techniques are rapidly becoming inadequate as software systems grow larger and increasingly more complex. Also, a consensus is building that the new paradigm of object orientation may help control complexity and harness the expanding system environment into more useful and exciting applications.

Applications will need to satisfy more sophisticated requirements, use more complex data structures and architectures, and be delivered to an increasingly broad base of users. Software developers will have to increase their capacity to build, extend, and maintain complex, large-scale systems including their existing legacy systems. This requires that software be more flexible and easier to use.

Much of today's software development processes are out-of-date, with programmers still functioning like craftsmen. They build unique, noninterchangeable components and assemble them by hand, and then they struggle over time to understand the code generated by their predecessors and to extend and refine that software. As powerful computers pervade the lives of more and more people, the inability to deliver and maintain equally powerful software is an increasingly visible problem.

WHAT IS OBJECT ORIENTATION?

There is no single, precise rule for describing or identifying object orientation. Rather, a collection of concepts together describe this new paradigm for software construction. In this new paradigm, objects and classes are the building blocks, while methods, messages, and inheritance produce the primary mechanisms. Historically, creating a software program involved creating processes that act on a separate set of data. Object orientation changes the focus of the programming process from procedures to objects. Objects are self-contained modules that include both the data and the procedures that act on that data. The procedures contained within the object take on a new name, methods. Objects are activated by messages. Objects that have a common use are grouped together in a class, and new classes can be created that inherit the procedures and data from classes already built. This inheritance enables the programmer to reuse existing classes and to program only the differences. This provides for a new level of abstraction, with prebuilt libraries of classes and even prebuilt application specific class libraries or frameworks. Object orientation is important for the software development challenges of the 90's. This paradigm will improve the software development process and will cause new and better applications to evolve. It's promises will be delivered incrementally and across a broad range of technologies and will permeate the next generation of software architectures.

BASIC MECHANISMS

OBJECTS

Webster's New Collegiate Dictionary defines an object as "Something that is capable of being seen, touched or otherwise sensed". Grady Booch, in his book, "Object-Oriented Design with Applications", defines an object as "Something you can do things to. An object has state, behavior, and identity; the structure and behavior of similiar objects are defined in their common class". David Taylor, in his book "Object-Oriented Technology: A Managers Guide" defines an object as "A software packet containing a collection of related data and methods for operating on that data".

Within objects reside the data of conventional computing languages, such as numbers, arrays, strings and records, as well as any functions, instructions, or subroutines that operate on them.

MESSAGES

Objects have the ability to act. Action occurs when an object receives a message, that is, a request, asking the object to behave in some way. When object-oriented programs execute, objects are receiving, interpreting, and responding to messages from other objects.

METHODS

Procedures called methods reside in the object and determine how the object acts when it receives a message. Methods may also send messages to other objects requesting action or information.

CLASS

Many different objects may act in very similiar ways. A class is a description of a set of nearly identical objects. It is a category or collection of objects that share a common structure and a common behavior but contain different data.

INSTANCE

An instance is a term used to refer to an object that is a member of a class. Instance and object are used interchangeably.

INHERITANCE

Inheritance is the mechanism for automatically sharing methods and data among classes, subclasses, and objects. A powerful mechanism whereby classes can make use of the methods and variables defined in all classes above them on their branch of the hierarchy. Inheritance allows programmers to program only what is different from previously defined classes.

KEY CONCEPTS

ENCAPSULATION

Encapsulation is the process of hiding all of the details of an object such as its data (instance variables) and procedures (methods). This is also referred to as "information hiding".

ABSTRACTION

Abstraction is the process of creating a "superclass" by extracting common qualities or general characteristics from more specific classes or objects. Each level of abstraction makes the job of programming easier because it makes more reuseable code available.

PERSISTENCE

Persistence refers to the permanence of an object, that is, the amount of time for which it is allocated space and remains accessible in the computer's memory. The object may continue to exist even after its creator ceases to exist. Objects stored permanently are termed persistent.

POLYMORPHISM

Objects act in response to the messages they receive. The same message can result in completely different actions when received by different objects. This phenomenon is referred to as polymorphism.

OBJECT-ORIENTED APPLICATIONS

Object-oriented applications will inspire users to think differently about the nature of computing. Programs in an object-oriented environment will be transparent. Object-oriented frameworks will facilitate simulating and constructing user-specific solutions. Objects will be shared in networking environments to distribute information within a work group or to parcel out tasks for distributed processing.

Object orientation is favored for applications which are characterized by complex processes and complex data manipulation. Applications in the following categories are classic candidates for enhancement through object orientation:

* Computer Aided Software Engineering (CASE)
* Computer Aided Instruction (CAI)
* Computer Integrated Manufacturing (CIM)
* Computer Aided Publishing (CAP)
* CAD/CAM/CAE Systems
* Document Management Systems
* Executive Information Systems
* Geographic Information Systems
* Graphics, Handling ICONS
* Health Care
* Image Storage Management
* Knowledge Based Systems

* Multimedia
* Manufacturing Production Control
* Manufacturing Requirements Planning
* Military Command and Control Decision Support
* Network Management
* Real Estate
* Systems Configuration and Version Management
* Telecommunications Routing Systems
* Visual Programming

Object-oriented applications will most likely gain in both presence and popularity.

POTENTIAL BENEFITS

Before managers can make informed decisions about adopting a new technology, the advantages of this technology must be translated into measureable benefits. Object-oriented programming improves not only the software development process but also the flexibility and utility of the resulting software. The design process becomes more intuitive as elements of the software correspond to elements in the application's real world domain. The programming process itself encourages teamwork, code reuse, and code polishing.

Reusability is the key to increasing productivity in the face of increasing complexity. The key breakthrough in object technology is the ability to build large programs from lots of small, prefabricated ones. In addition to the increased productivity that results from reusability, using object-oriented technology can result in greater reliability because it reduces the risk of human error. Program structures remain intact, and change propagates naturally through the hierarchy of classes.

Flexibility is also a trademark of object orientation. Programmers are freed from the constraints of preestablished data types, allowing extensions of application functionality and bridging of heterogeneous applications.

Adaptability of object-oriented programs may well turn out to be the most crucial advantage of object-oriented technology. No matter how perfectly crafted, a program is useless if it doesn't meet current needs and the needs of users are changing at an ever increasing rate.

Faster development of applications is another benefit and is a result of all the programming effort that is reused from existing objects and all the design work that went into an existing model of a process.

Increased scalability is another significant benefit of object orientation. Given its improved modularization, it is especially well suited to developing large-scale systems.

Large systems are easier to build and maintain when you build them out of subsystems that can be developed and tested independently.

POTENTIAL CONCERNS

While object-oriented technology promises many benefits, there are some valid concerns about its ability to deliver those benefits. Most of these concerns have to do with temporary limitations and should disappear as the technology and its market mature.

The maturity of the technology itself is a concern to many potential developers. It is not yet a completely stable technology and many companies are not comfortable being "pioneers".

Standards are still evolving and the lack of accepted standards raises concerns about the difficulty of moving programs from one development environment to another and mixing and matching objects and classes from different vendors. Standards are on the way. The Object Management Group (OMG) was formed by a consortium of the major vendors of several object-oriented products. The purpose of this group is to promote the adoption of standards and the interchangeability of objects. In addition, the American National Standards Institute (ANSI) has an ODBMS committee, but no standards have been officially approved.

There is also a shortage of tools for application development. These tools include programs to assist in the design of objects and the management of libraries of reusable objects.

Performance of object-oriented applications is a concern and the speed of object-oriented systems will improve as the technology matures.

The object-oriented approach has a tremendous amount of potential and companies should explore this new technology, and check out the benefits for themselves.

EVOLUTION OR REVOLUTION?

Many organizations have invested heavily in existing non-relational and relational database management technology. The majority of these companies do not want to replace their existing databases and applications. The need to integrate these "legacy" databases with each other and with new systems is an important factor in the future evolution of data management.

There are clearly risks associated with getting into this technology too soon. But there are risks associated with waiting as well. The companies that begin the transition now will enjoy an important competitive advantage while the others strive to catch up.

The most prudent strategy is to avoid the extremes of ignoring the strategy or committing vital systems to it. Instead, companies can make a modest investment in a pilot program to gain first-hand experience with object-oriented developmenmt. This approach allows a company to reach its own conclusions about the value of the technology, and places the company well down the experience curve if it converts in the future.

THE FUTURE

Object-oriented technology will provide the clarity and flexibility essential to the successful development of complex systems. Today's applications do not offer the consistency and flexibility needed to make the computing environment more productive for users. Object orientation will provide environments in which users can communicate among applications and navigate easily over distributed, heterogeneous architectures.

In the near future object orientation will deliver the most benefits to three categories of programmers: power users, general business programmers and system developers. The most dramatic near-term benefits will be for system developers who both require and embrace this development approach and evolving tools to implement the increasingly complex and potentially innovative software of the 1990's. Over time, object-oriented technology will begin to have increasing impact on general business programmers and power users. Carefully designed object libraries will become available to support less sophisticated programmers who want to assemble applications quickly from prefabricated objects.

The vision of the future extends beyond the arrival of object-oriented system components, development tools and standards. In the future, users will have the power and flexibility to design their own applications just by snapping together the necessary objects. With objects, building applications will be a process of tailoring and linking reusable modules. Object-oriented software architectures will mature in the 1990's. The transition to these new architectures is underway, marked by the arrival of object-oriented languages, databases, interfaces, operating systems and development environments. New types of data, distributed processing, multimedia applications, and end user computing are driving forces in the implementation of the object-oriented software environment.

There has been a great deal of progress in implementing object-oriented systems. Today's graphical user interfaces have acquainted users with object manipulation. Among object-oriented languages, C++ has become the de facto standard. Object-oriented extensions are also being implemented in most popular commercial languages. Object-oriented development environments such as Hewlett-Packard's Softbench provide examples of object-oriented programming and applications. Operating systems are also being extended to support interoperability among object-based applications.

Over the next decade, the difference between the old and the new will become increasingly obvious to both programmers and users. When the object-oriented future is fully delivered, this natural, intuitive paradigm will be strongly embraced and will provide benefits to programmers and users alike.

OBJECT TECHNOLOGY FROM HEWLETT-PACKARD

Hewlett-Packard recently announced HP OpenODB, the most advanced, commercial object-oriented database management system for large, multi-user environments. It is designed to enable new, complex business applications to be developed and maintained at a fraction of current costs.

HP OpenODB is based upon the Iris ODBMS prototype developed by HP Labs, starting in 1984. Using Iris, HP has worked extensively with customers and universities in evaluating ODBMS requirements. HP OpenODB is targeted at complex, commercial applications such as Geographic Information Systems (GIS), telecommunications systems and heterogenous information integration such as Executive Information Systems and Computer Integrated Manufacturing, whose needs may not be met by current database products such as TurboIMAGE and ALLBASE/SQL.

HP OpenODB uses a relational database as its storage manager and presents an object-oriented model to developers. It is a hybrid approach which allows integration of existing legacy systems including applications written in C, COBOL, FORTRAN, PASCAL, ADA, etc. This architecture provides a robust storage environment with the DBMS capabilities commercial users have come to expect. HP is currently using ALLBASE/SQL as the storage manager for HP OpenODB for performance and stability of data; however, the architecture allows HP OpenODB to be ported to other relational DBMS's for portability to non-HP hardware. This combination is unique in the industry.

A Key benefit of HP OpenODB is that users don't have to abandon their current software or data to work with it. It includes an object-oriented structured query language (OSQL) and external functions that will retrieve information regardless of it's format or whether it is stored inside or outside of HP OpenODB.

SUMMARY

Object orientation is another phase in the evolution of computing and is an important step towards the vision of "Information At Your Fingertips". Developers must take advantage of the many benefits of this technology while at the same time deal with the hurdles that this technology poses. Object-oriented development environments must play a large role in reducing the learning curve and make object-oriented programming a highly productive process. Object-oriented products are here today and the commercialization of object-oriented technology is increasing rapidly. Object based architectures lend themselves to the creation of a much richer information environment. Digitized voice, music, video clips and animation will begin to populate our information systems. Object database systems are currently viable for commercial projects and will be widely adopted by the mid to late 1990's.

John Podkomorski
Hewlett Packard Company
1266 Kifer Road
Sunnyvale, CA 94086

Information Technology and Management Information System (IT/MIS) management is becoming more complex as a direct result of industry forces, technology, and the changing expectations of management, users, and staff. These changes are often subtle, but have a very significant, and often confusing, effect on the day to day management requirements of the IT/MIS management team.

This session develops a model to help understand the changes which affect the IT/MIS environment. Historical pressures will be compared to new pressures for the decade of the 1990's. In addition, a group discussion will focus on possible strategies for managing the new set of pressures felt by IT/MIS managers.

AUDIENCE:

All levels of IT/MIS management, especially department management, and the management and staff of application development , systems design and implementation, and technology support.

A TIME OF RAPID CHANGE

The IT/MIS management world is becoming much more complex. Change has become so very rapid that it often feels as if the comfortable patterns of the past are all changing at the same time.

Where will IT/MIS managers feel pressure through the 1990's? How are the pressures changing? And why?

There are two basic sources of increased pressure on IT/MIS Management:

1. Technological development, and industry trends.
2. Organizational pressures from user communities inside each company.
   - A. Upper Management
   - B. Users of IT/MIS services
   - C. IT/MIS Staff

This paper will discuss the changes in each of these very important areas.

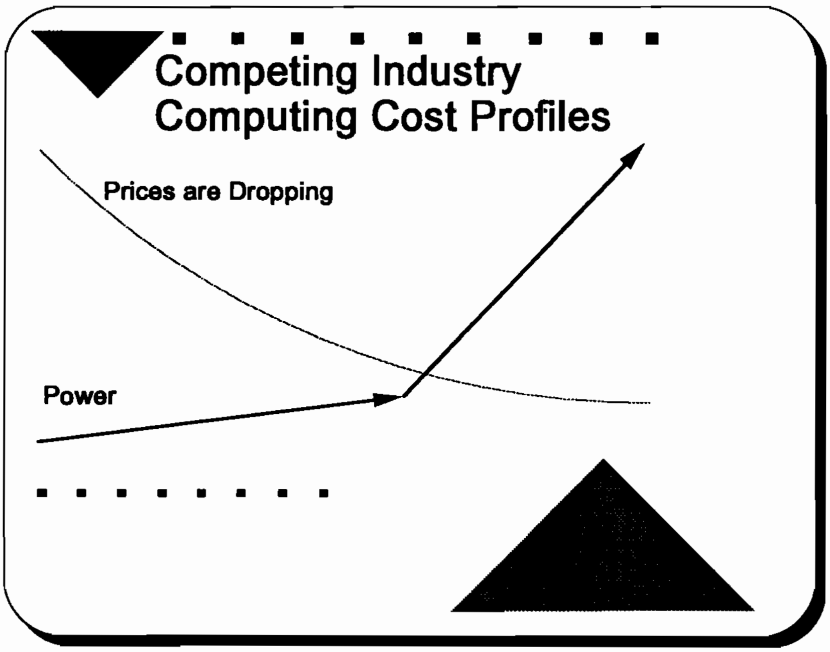
This model is not really very creative. These factors have been the major driving force behind IT/MIS for many years. What is changing rapidly, however, is the effect of industry developments and technology progress, and as we'll see later, the set of expectations each of the three organizational "users" now have for IT/MIS.

First, let's look at the effect of industry and technology changes.

This is the area which is the strongest contributor to the rapid change in the IT/MIS environment. Technology change is happening at an unprecedented pace, throughout the industry. At the same time that technology is changing rapidly, we've seen that the forces of business changes are moving equally rapidly, with unprecedented effects on competitive pricing. The combination of these two factors amplifies the effects of both.

Let's look at just a few of the industry issues IT/MIS needs to deal with.

PRICES ARE DROPPING



Computer price performance curves are changing dramatically. Some time ago, a computer vendor company was considered aggressively successful if it could maintain a 20% per year improvement in price performance. New computer models were released to market perhaps once every two years, and a 5 - 8 year expected life cycle was not uncommon. There was a clearly visible difference between the "power" of a mini-computer, and the "power" of a mainframe.

PERFORMANCE IS INCREASING DRAMATICALLY

As a direct result of RISC Architectures, today's vendors are increasing the power in their processors at nearly 100% per year. Just 4 years ago this rate of increase was respectable at 20% per year. It is not unusual to see very rapid announcements of new computer lines which totally replace the "state of the art" just 9-12 months back.

The historical gap between "mini-computers" and "mainframes" is not as clear today In today's market, there really are alternatives to the mainframes of the past. MIPS, and very large disk and network configurations, are readily available from very high power processors, often fully software compatible with the distributed processors currently in place. "Mini-computers" supporting many hundreds, and soon to be thousands, of users are common. The management of these computers requires different resource algorithms than those used on minicomputers in the recent past.

One expectation which has not changed, however, is that the operation of these more powerful computers will be as easy, or preferably easier, than the operation of previously distributed mini-computers, and will certainly never be as complex or expensive as managing a mainframe environment.

PURCHASE DECISIONS ARE HARDER

Purchase decisions are increasingly difficult as a result. IT/MIS must ask two questions, and balance the answers carefully...

> What if I buy today, and a newer, more powerful, cheaper model is released in 3 months? I'll really look silly!!!
>
> What if I don't buy today, and my need for increased computing causes a business problem for my company? I'll really look silly!!!.

ECONOMY OF SCALE

In an interesting turn of events, the significant increase in the power of mini-computers is causing many companies to rethink their decisions to "decentralize". In very large corporations, there is an increasing trend toward increasing the size of computing facilities, with a strong emphasis on economy of scale and improved IT/MIS operational efficiency.

This trend, however, is quite different from a return to mainframes. Companies really want the ease of use and operational simplicity of a mini-computer environment, while also getting the advantage of economies of scale in larger operations.

SOME SIDE EFFECTS:

There are several side effects caused by larger organizations, including financial factors, new management skills, new vertically integrated work groups , and a significantly higher dependance on data communications technology.

FINANCIAL FACTORS:

Historically, many, if not most, decisions in IT/MIS were left to the technical discretion of the IT/MIS manager. Computing technology was not commonly understood by management, and as a result the research, judgment, and decision making was left largely to the IT/MIS community. Distributed operations in the past several years were justified by their immediate customers, and accounting, or chargeback systems, were not important.

This is changing today, due mostly to the increasing pressure on financial performance, and considerably higher attention by "upper" management. As more technologically trained managers rise through the ranks, financial pressure is often reflected onto technical decisions. The chargeback systems come under close scrutiny. Service level agreements need to be negotiated between managers.

IT/ MIS managers no longer get to say "Trust me - its a technical thing!!".

NEW MANAGEMENT SKILLS:

Minicomputer IT/MIS organizations of the recent past have been proportionately small, especially when compared to "typical" mainframe shops. Performance or capacity planning was proportionately simple, with many processors managing a single workload, or a limited combination of focussed workloads. As more powerful computers are implemented, IT/MIS management must implement many of the techniques formerly only used on mainframes. Increased capacity planning and forecasting is critical. Accurate and representative chargeback systems are suddenly necessary. IT/MIS is evolving from a technical service to a "business function".

Also, the customers of IT/MIS are becoming more computer literate, and as a result are demanding more comprehensive and comprehensible data about the operations of their computers. Performance and billing metrics are increasingly necessary, stated in business terms which are understandable by end users.

VERTICAL WORK GROUPS:

There seems to be a current trend in industry emphasizing flatter organization charts with fewer layers of management. This style of organization has a much different decision making model, especially compared to a deeper, more monolithic structure. There is a higher need for rapid, effective, and efficient information dissemination, as well as a requirement for flexibility, and responsiveness to rapid change.

In smaller, often distributed, organizations it is common to have a very "general" work force, with fewer employees managing a processor, each with a very broad set of skills. As organizations get larger, it is increasingly necessary to develop technology specialists, with less breadth in their assignments, and considerably more depth.

DATA COMMUNICATIONS:

Larger processors also require a higher dependence on data communications capabilities. Some of the benefits of decentralization included a user's sense of control, with access at the user's desk, and listings and reports delivered quickly very close to the user's work place. As processors get larger, distances from users also get larger, and the need to distribute processing power (PC's, workstations, terminals), as well as remote printouts and reports, depends on data communications.

CHANGING EXPECTATIONS

IT/MIS managers will need to change their expectations, and develop new skills to match the evolution of the industry. IT/MIS managers need to increase their attention to the business factors in running their shops, aligning with the goals of their company, and closely accounting for the resources used.

There seems to be a current trend in industry emphasizing flatter organization charts with fewer layers of management. This style of organization has a much different decision making model, especially compared to a deeper, more monolithic structure. There is a higher need for rapid, effective, and efficient information dissemination, as well as a requirement for flexibility, and responsiveness to rapid change.

CHANGING VENDOR / CUSTOMER RELATIONSHIPS

We've seen earlier that prices for computing equipment are dropping rapidly, and that the power delivered by this less expensive equipment is increasing rapidly, resulting in extra-ordinary price performance for purchasers of computing power.

Unfortunately, the rapidly lowering prices are causing considerable stress in the funding algorithms used by most vendors. From a customer's perspective, things which once were "free" seem to be no longer "free". Most vendors are being forced into finding more efficient, less personal processes to sell their equipment, and no longer have the freedom to include many of the "extras" we've come to know and love.

STANDARDS

And then there is the "open systems" question. What is the "right" thing to do in an IT/MIS department? How do we decide whether to make changes across vendor boundaries, or Operating System boundaries? What criteria should we use? If we stay proprietary, will that mean we're "trapped"?

Can your company afford to do the development necessary to re-implement all applications on an open systems platform? Can your company afford retaining the status quo?

How do I decide whether it is better to stay proprietary, choose a UNIX solution, or choose a POSIX compliant solution? What is the "best" road?

Will our staff continue to work for us if we don't go to UNIX? Will our staff continue to work for us if we do go to UNIX?

APPLICATION DEVELOPMENT TECHNOLOGY

Another contributing factor is the development of new technology in application development methodology.

The recent entry of computer engineering (CASE) products into the marketplace is changing the expectations of IT/MIS staffs, as well as those of our users. CASE tools are reported to make development much faster, and with "perfect" results. Developers and designers are beginning to demand access to CASE technology (perhaps in self-defense) in order to increase their own productivity.

4GL's have also increased our user's expectations about the speed with which applications can be delivered. 4GL's promise development times considerably lower than historical methods.

4GL's have another interesting effect. As computer literacy increases in the user base, more and more users are feeling competent in inventing or modifying their own applications. IT/MIS has considerably less control in an environment where users can write inquiries, reports, and even update their own records using 4GL technology.

The advent of relational data bases, and SQL access methods, amplifies this trend. As the elements of a company's data become easier to manage, and more intuitive to access, the nature of system design and implementation must change to allow for the considerably higher level of user freedom currently available. "Object" orientation is challenging the technology limits again.

PC INTEGRATION

Increased power in PC's and workstations, coupled with the software technology to utilize that power for user interface enhancements, and for file and data storage distribution is really changing the profiles of what new applications look like. The definition of "central" computing, or "distributed" computing, is getting pretty fuzzy.

Also, development of software packages - "shrink wrapped" software - is increasing our users' appetite for solutions. From a user's perspective, the software and applications marketplace infers that every business problem has a solution "on the shelf". All a company has to do is buy it, and all their problems will be solved. In some cases, that's true. In others, it is not very accurate.

Tough decisions everywhere...

Which is why you experience all the HYPE!!

HISTORICAL MANAGEMENT EXPECTATIONS

Let's look at the evolving expectations of Management. By management we mean the organization structure to which IT/MIS reports. Historically, this has been the Finance function in most companies, or perhaps the Manufacturing group, depending on the needs of each company. One change we see developing is the continued movement toward stand alone IT/MIS functions, at a par with other functions in the company. This change is one of the forces driving new sets of expectations.

AVAILABLE, BUT INVISIBLE

Historically, IT/MIS departments have been expected to be invisible, and have had the luxury of working in a limited monopoly situation. They had the technical experts, the equipment, the technology knowledge, and the endless jargon which kept the unruly users at bay.

Management expected the department to operate without complaints or surprises, and to manage their budgets with reasonable controls in place.

This expectation set has not changed significantly in the last 20 years or so.

NEW EXPECTATIONS

What *has* changed?

NEW MEMBERSHIP IN THE MANAGEMENT TEAM

The most significant change, which is made more visible by the movement to have separate IT/MIS functions in large companies, is the expectation that IT/MIS become more of a business partner with the other functions in the company. New demands are drawing MIS managers more into the management team to provide the "systems" expertise to solve business problems.
Current financial market conditions are forcing many companies to redesign business processes. IT/MIS organizations frequently have the professional skills to drive that redesign, and are increasingly being asked to do so.

Information management is increasingly expected to create a competitive advantage - to make a contribution to company revenue. Increased productivity, improved customer satisfaction, faster deliveries - all these contribute directly to company bottom lines, and all can be improved by more user-friendly and productive applications systems.

Instead of a "servant" to the management team, IT/MIS is now being asked to be an active partner, with considerable emphasis on the IT/MIS contribution to organizational development, profitability, and business process improvement. IT/MIS authority seems to be increasing, with an absolutely necessary shift away from the comfortable ground of technology to the much more slippery arena of business management. This often brings along an increase in authority for

IT/MIS management, but not necessarily with the enthusiastic endorsement of the rest of the management team. The "politics" have changed.

Recent press articles, however, have begun to suggest that upper management is not satisfied with current MIS contributions.

EMPHASIS ON FINANCES INSTEAD OF TECHNOLOGY

Another change in the dynamics of companies is the increased assertiveness of the finance function over technical decisions once tightly held in IT/MIS. Because of increased technical exposure, many more managers are becoming remarkably competent at asking very hard questions about the economics of IT/MIS development and operations. We have seen real cases where a change as radical as a shift in vendors has been caused by a directed budgetary limitation.

> "If finance says we must operate for $1M less per year, how can we do that? We may need to switch vendors!!".

GLOBAL REQUIREMENTS

Global influence is rapidly rising. Companies can be "national", "bi-national", "multi-national", or "global". Because of the shrinking size of the world, many more companies are seeking opportunities to enter the global marketplace. The world market brings along many very interesting management challenges, and a whole plethora of technology challenges, which were not commonly considered even 10 years ago.

U.S. companies, in particular, are unaccustomed to the stresses of international or global business. This increasing "global" requirement represents a new type of thought. Cultural issues, as well as the complexity of international finance, standards, and inter-country laws must be considered as an active part of "system implementation" decisions.

Here are some results from a recent survey done by the INDEX GROUP for 1991. The survey participants were a group of 394 U.S. and Canadian I/S executives, with companies greater than $250 million. 74% of these companies had revenues over $1 billion.

The chart includes the results from the last four years. It is very interesting to look at the rate of change in many of the items, as well as the number of items which were not on the list even two years ago, and have risen greatly in importance.

Let me draw your attention to some specific items.

#1 on the list is "Reshaping Business Processes through Information Technology". In 1988, this was not thought to be a major issue. For the last two years, however, it has been #1.

#10 on the list, "Enhancing Leadership", has not been mentioned before. In 1991, however, it has risen to #10. This is very key to the success of IT/MIS organizations. Upper management expects much higher leadership than it has in the past. At the same time, upper management is not sure that current IT/MIS management has the leadership skills to deliver against those expectations.

#4, and #7 on the list specifically address 1991's emphasis on the highly productive development of high quality software. It is easy to see the increased emphasis placed on productivity and quality.

1991 **INDEX GROUP** SURVEY RESULTS

1988 Rank	1989 Rank	1990 Rank	1991 Rank	DESCRIPTION
None	11	1	**=>1**	**Reshaping Business Processes through Information Technology**
1	2	4	2	Aligning Information Systems with Corporate Goals
None	7	3	3	Instituting Cross Functional Systems
12	13	6	**=>4**	**Boosting S/W Development Productivity**
7	6	7	5	Utilizing Data so that information is accessible and used by the right person at the right time
2	4	5	6	Developing an Information Systems Strategic Plan
None	Non e	14	**=>7**	**Improving Software Development Quality**
5	5	9	8	Creating an Information Systems Architecture
6	12	16	9	Integrating Information systems, solving technical problems of integrating heterogeneous h/w and s/w
None	Non e	None	**=>10**	**Enhancing Leadership Skills at or near the top of the IT/MIS Organization**
17	14	10	11	Cutting IT/MIS Costs
4	1	8	12	Using IT/MIS for Competitive Breakthroughs
8	8	11	13	Improving the IT/MIS human resource
3	3	2	14	Educating Management on IT/MIS
None	Non e	19	15	Connecting to Customers and Suppliers
9	10	17	16	Managing Changes caused by IT/MIS
None	Non e	15	17	Promoting the IT/MIS Function
None	20	23	18	Determining the value of IT/MIS
13	16	25	19	Managing dispersed Systems
None	17	21	20	Capitalizing on advances in Technology

As you can see, the list of top 20 needs of the IT/MIS have shifted a little over the last four years, with management needs expanding, and technology needs dropping a bit.

Let's shift, now, and look at how the expectations of USERS of IT/MIS are changing.

HISTORICAL USER EXPECTATIONS

"Users", in this context, are the departments in a company who actually receive the benefit of IT/MIS development and operations work.

Historically, users have expected solid, reliable, available, cost effective services from IT/MIS. This, too, has not really changed in 20 years. Users have always been concerned about their application development backlog, but have grudgingly accepted it because of the perceived difficulty of implementation of complex IT/MIS programming projects.

In historical IT/MIS installations, user personnel would operate a single application, often heads-down to the keyboard, for hours on end. They were specially trained for that application, and worked with it for a long time. Specific application functions needed to be well tuned, but users did not need to be very versatile, and it was acceptable to use training to overcome some system shortcomings.

IT/MIS provided a service, crunching numbers and inventory records, and keeping track of historical data better than people could.

How are users' expectations changing?

TODAY'S USER EXPECTATIONS

User departments are drowning in the combination of budget restrictions, personnel shortages, labor shortages in key skills, and an increasingly demanding business environment.

PRODUCTIVITY, AND MORE PRODUCTIVITY

Because of cost pressures, there is a very significant emphasis on productivity in user departments. User personnel are being asked to operate efficiently in multiple applications. User departments expect their staff to be multifaceted, and fully knowledgeable in multiple, unrelated, applications like word processing, payroll, customer management, phone inquiries, and electronic mail.

Users read every day about decreased training time, enhanced context sensitive help facilities, intuitive user interfaces, and "push button" instantaneous context switches, and they are rapidly expecting IT/MIS to implement these more powerful environments into their everyday lives.

There is a dramatically increasing need for user proficiency, with a corresponding desire for reduced requirement for training. Modern training is expected to be painless, blindingly quick, and to be self-paced, without a required instructor.

The productivity of user personnel is now affected not just by training, or by experience, but by the complexity of the multiple application environments, and by the time it takes to switch contexts between them. One HP customer calculated that context switching alone - the time to switch from one application to another - was costing over $2M each year.

USER PARTICIPATION

In addition, more computer literate users are demanding a much stronger role in the design and development of their own applications. Many of these same users feel empowered to develop some applications on their own. This pattern places new training, documentation, and support requirements on the IT/MIS function.

Users no longer want single focus applications which require that they learn the quirks of the application. They want a user interface to be knowledge based, intuitive, and adjustable to match their behavior. They want their applications to blend naturally into all the work of the company.

MAKE IT, OR BUY IT?

Another subtle pressure is the "make / buy" decision. In the '60's, we "made" everything. We had to, because there was a limited 3rd party market. In the late 70's and early 80's, IT/MIS began suggesting the possibility of purchasing packages, and modifying them. This "buy" decision was often an opportunity to save a lot of money by eliminating the original invention cost. Unfortunately, it was nearly as often the cause of extensive budget over runs, when modifications cost more than an original implementation might have.

Times have changed with the development of a very viable 3rd party market. There are many more applications on the market, many of which are very directly applicable to common business problems.

Our users are rapidly embracing the perception that "buy" will work - "It says so right here in the paper, so it must be true". This puts IT/MIS in a very delicate position. If IT/MIS suggests a "make" decision, users ask why it can't be bought cheaper. If IT/MIS suggests a "buy" decision, users ask why we need such a big IT/MIS department, and "Is it installed yet and running exactly right, and why not?"

NEW DEFINITION OF QUALITY

In addition to all this, users are fully expecting unprecedented quality levels - much higher than in the past. "Quality" has been talked about, and written about, a great deal in the press, and in professional circles. Users now expect very few problems with new systems or applications and are very intolerant of down time or interruptions. As more users become technically "computer literate", this expectation will very likely increase. Budget and staff reductions in user organizations make outages much more expensive to the users, and any data loss is critical.

COLLABORATION

IT/MIS is increasingly expected to leave its comfortable role as a service provider, and enter a newer, more complex collaborative team player. This newer role requires much more than IT/MIS technology skills; organizational development, finance, business judgment, and customer management are increasingly important.

Let's look at how staffing issues are changing.

HISTORICAL EXPECTATIONS OF IT/MIS STAFF

IT/MIS staff have always been an unruly lot. Because they have had special knowledge they have demanded, and have usually received, some special treatment.

Historically, IT/MIS staff has expected a "good" job, with lots of variety, job enrichment opportunities, the ability to develop depth or breadth (personal preference), and growth potential.

Programmer analysts were competent if they had a good working understanding of COBOL, Screen management, and a data base. Specifications were usually technically developed, and given to programmers to implement.

The historical focus on staff management was retention of your best employees. In many cases, recruiting could be easy, depending on your environment and how much you were willing to pay.

What's changed here?

TODAY'S STAFF EXPECTATIONS

The major change has been the market's reaction to the decreasing numbers of qualified technical staff. Management focus, in addition to retention, must now be on recruiting requirements. In some ways, IT/MIS management must continuously recruit its current employees.

In the past, personal progress could have been measured by promotion, or by technical growth. Because of more vertically aligned organizations with flatter organization charts and fewer managers, the promotion path is not as available today. There are simply less managers, so there are less chances for promotion.

On the technical growth side, developers, programmers, programmer analysts now demand to work "at the state of the art". They ask for strong, nurturing environments, with superb tool sets, increased training, and skills development plans.

Tomorrow's technical staff will require much more than language, terminal, and data base knowledge. They will need sound business judgment, consultative skills, and a broad knowledge of all aspects of the business enterprise. With user departments capable of doing some of their own programming, the type of technical contribution made by IT/MIS staff will change a lot in the near future.

This feels quite similar to the needs of the past, but is different in one very important way. The decreasing numbers of qualified job applicants is making the job environment very competitive.

A different type of management style is increasingly required to attract, manage, and retain qualified technical talent. More focus on individual needs, job enrichment, personal growth, and the whole working environment is required today.

PRESSURE EVERYWHERE

Which brings us to the IT/MIS situation for the 90's. Pressure, pressure everywhere, and not much relief in sight.

IT/MIS management must keep track of a dazzling array of technology changes, understand what they really mean to the company, measured in business terms, and become very effective at sharing that knowledge with the rest of the company. IT/MIS departments will need to keep up with technological change, ***and*** will need to learn to teach the value of that technology to their management partners, in ***business*** terms instead of technological advantages.

IT/MIS management is increasingly expected to play a stronger role in the corporate management team, making a direct contribution to the business. We must rise above the technology, develop strong management skills in business issues, build departments which have a strong positive effect on our company's bottom line, and learn to be a partner in our management team. Management of global issues will be continuously more important as this decade progresses.

IT/MIS management is increasingly also expected to collaborate with its users, no longer providing services, but pro-actively improving productivity in a rapidly changing, complex application environment. Improvements in the quality of IT/MIS deliverables will be a key success factor.

IT/MIS management must react to a growing shortage of technical staff, with the resulting need to really focus on staffing issues. Movement to modern development techniques, use of recent design and implementation tools, and use of relational databases will provide an exciting environment. Staff will demand more training, and new kinds of growth potential as organizations get flatter. Lots of creativity will be required here.

More communications skills are required today than have ever been required in the past. Negotiation skills, listening skills, and even the ability to teach will be very important.

More business skills, and business systems skills are required than have ever been necessary before. This will be especially true as IT/MIS managers are integrated into the mainline of the company management team.

In summary, IT/MIS must look forward, and develop new skills very quickly, to deliver on the expectations of upper management.

A shift from technology management to business management, and a new way of contributing to the management team, is increasingly important.

Tracking technological change is probably the largest current challenge. Technological change is happening so quickly that it is very difficult to track, and even more difficult to make "correct" decisions. Management wants decisions made on a "long term" basis, with predictable stable results. The technology world seems to be changing under our feet about every 9 months or so.

IT/MIS will need to do more training of its users, and will need to shift its role from service provider to technology mentor. Controls on user data will be minimized, and user access to data will be unprecedented.

And all this must be done in chronically tight fiscal situations, with an increasing shortage of qualified technical personnel.

The computer marketplace is incredibly volatile. Change is everywhere.

It is all happening at once, on many different axes, in many product areas, with global implications.

It is all happening with unprecedented speed.

And we are all in the middle of it!!! What an exciting time!!!

Paper# 3857
Implementing a Client/Server Application

Stayton Chock
Hewlett-Packard Company
Corporate PeopleBase Program
3000 Hanover Road, MS 20BBD
Palo Alto, California 94304

(415) 857-4814

Introduction

In 1990, Hewlett-Packard (HP) Corporate's PeopleBase program embarked on a systems planning project, which resulted in the development of a systems concept for HP's U.S. people management systems. This concept was based on a client/server architecture with a centralized server and remote clients throughout the U.S. This was a major departure from the 20 year old flat file payroll system and distributed human resources systems, which used traditional COBOL, Image, and Vplus. In order to do this great undertaking, anything that would improve the schedule and reduce the overall cost to the program was considered. PeopleBase chose a third party product called PeopleSoft using a Vectra PC (an IBM PC compatible) running MicroSoft Windows as the client and an HP 3000 system running Allbase as the server.

Problem Statement

The following problems with the current system made the payroll and human resources community realize that they needed a new system:

The current payroll system is on a mainframe, which costs a significant amount of money to upkeep each year. In addition to that, it is difficult to make changes as the "database" is implemented on a large flat file. Although this system is old technology, payroll runs are very fast because one pass through a large flat file is faster than going through the overhead of indices and hash tables on a SQL database.

The human resources systems had problems of having little or no history, missing functionality, few edits for data input, and data being decentralized. This means that when people transferred from one division to another, employee data would have to be manually typed into the receiving division's system. There were many times when that retyping was not done, which resulted in a loss of work history. Also, with HP's many reorganizations, divisional boundaries have changed frequently and getting different cuts of human resources data has become more difficult. When group managers or sector managers (managers above division management level) want a cut at their data, decentralized systems make that reporting more difficult.

Choosing a Package - PeopleSoft and Client/Server

When it came to choosing PeopleSoft as our client/server package, there were some definite benefits and risks that needed to be weighed. Doing client/server was a leap for this area, but purchasing a package was a further leap. The information technology area made a major push toward using packages. But since the current 20 year old payroll system was home-grown and HP had a history of using home-grown systems, there was a strong tendency for the next system to also be home-grown.

Benefits

- Rich functionality

  The vendor has several full time functional experts working to provide rich functionality in the product and to maintain a rich vision for the direction of the product. In addition, this helps reduce the risk of failure a custom development project could experience based on the inability to design a fully functional people management system.

- Good vanilla demo product

  The demo product is a very functional human resources and payroll system for a fictitious bank with rather comprehensive data for 125 people. By using the system for a few minutes, a user can see the breadth and depth of functionality of PeopleSoft. It satisfied many of HP's business needs. But how about the others?

- Strong tools for customization

  One of the strongest points of the product is its tools for customization. PeopleSoft tools include a record editor to create and modify records (SQL tables) in the SQL database, a panel editor that takes fields from the record editor and places them on a screen as defined by the user, a menu editor that takes panels and arranges them into any number of menu configurations, a help editor that customizes help on a field or panel basis, and "People-

Code" that allows the developer to edit fields on input, do processing, or even modify the SQL database.

This "4GL" (short for 4th generation language; usually a language with more powerful macros than ones that come in standard programming languages, but used only for a specialized area) called PeopleCode is specific for panel and record needs and cuts down on the amount of code that would be needed, if the code with the same functionality were written in C. In addition to that, certain reports are written in SQR, a 4GL language that the package vendor uses. Other than that, batch jobs and some reports are written using C for performance reasons. PeopleBase was very careful not to choose today's best 4GL because in the past, 4GL's would be superb one year and unacceptable the next year. This could be because they wouldn't keep up with database enhancements or because business was not favorable for them and as a result, their customer support would dwindle.

Since PeopleSoft's product had strong customization tools, PeopleBase was choosing a software package that would not be easily outdated, as business rules and/or functionality changed.

- Improving the schedule (time to market for an internal product)

  If a package did exist, fit the parameters of the business, and had a high enough level of quality and support, time to market would be significantly shorter than a system that had to be developed from the beginning. Strong customization tools should also help drive down development time.

- Lower cost

  A good package means that less development is required and that translates to lower costs. A smaller support team is also needed. This means that more resources can be concentrated on data conversion.

- Leverage vendor/other vendor customer experiences

  So far, PeopleSoft has hosted two user conferences, where customers can get together and compare notes to see if one company can leverage off experiences from other companies. Sharing data in this forum doesn't seem to be seen as sharing internal company secrets. Several companies have gotten together to form their own special interest groups to lobby PeopleSoft to make enhancements.

- Several other large companies leaning toward PeopleSoft

One of the big questions that PeopleBase had about PeopleSoft was whether the product would work for large companies. As is, the vanilla product looked good for small companies. At the time the product was selected, there were no large company successes, but several other notable large companies were leaning toward PeopleSoft. As far as it is known today, there are at least three notable, large companies in development.

- No close competitor

  Some of the criteria for a prospective package product were that it had strong functionality for human resources and payroll. In addition, there were several other hardware and software platform wants for the package including: (1) be true client/server, (2) use open systems, (3) have tools for customization, and (4) use HP equipment. After surveying the market, PeopleSoft was by far the best choice.

- Client/server using SQL based databases

  Implementing client/server has been a Hoshin goal for several divisions for a few years. As a rule, Corporate uses new technology developed by HP's manufacturing divisions. Although using COBOL/Image/Vplus has been the norm for developing business applications in Hewlett-Packard, at the time the package was selected, there was a major push by the director of information technology to utilize client/server applications and SQL databases. In this environment, this push was needed so that management could and would support this new and risky technology (leading edge compared to bleeding edge).

  As implemented by PeopleSoft, a client is an IBM PC compatible using MicroSoft Windows 3.0 connected to a network, which connects to a SQL database server. In PeopleBase's installation, we used a Hewlett-Packard Vectra QS20 or better, LAN Manager using Internet, and an HP 3000 Series 9x7 server running Allbase/SQL. It should be noted that the Allbase port took place during PeopleBase's investigation and design phases.

  Moving toward client/server technology was valuable because of the following factors:

  - Ease of use and widespread user interface

    MicroSoft Windows 3.0 and using the mouse are very easy to learn for the inexperienced user. With products such as Professor Windows, the novice can learn even more easily. MicroSoft Windows is also seen as one of the defacto standards for graphical user interfaces (GUI) and the leading standard for PCs. Software houses are more likely to write software for any standard (defacto or not) compared to them writing software for any non-standard.

- Potential future migration to Unix for a client and a server

  Since most of HP's business users have DOS rather than HP-UX (Unix) workstations and since there is far more business software on IBM PC compatibles than Unix workstations, using PCs was chosen over using HP-UX workstations. Today, PeopleSoft does not work on Unix workstations as clients and there are no plans to do so. This vendor believes that the market is not yet large enough to warrant work in this area. In the next few years, the technology for running PC software on Unix workstations should improve. There already have been a few PC emulation products on the market, but all of them are believed to have poor performance. As for the server, using the HP 3000 Allbase server platform allows the flexibility for a migration path to the HP 9000 server in the future.

- Interactive performance requires mostly network and client horsepower

  One of the strengths of client/server is that the client can take a disproportionate amount of the CPU burden. The server should not be the bottleneck, because if it is, a very limited number of clients can be connected. PeopleSoft is an application, where performance goes up significantly when a faster PC client is used (The increase is somewhat proportional to the increase in the megahertz rating of the faster PC to the slower PC).

  The greatest concern has been in network performance. PeopleSoft doesn't keep edit tables on the client so every time a field is entered, it may have to go across the network to see if that field is on a table and if it is valid. To help address this network performance issue, PeopleSoft created the translate table, which is a special type of edit table cached on the PC.

  Performance specialists have been working toward reducing the number of messages between the client and the server. A part of this work was completed by collapsing messages so that compile, fetch and execute (3 messages) can become one message. Significant other work has also been done.

- Can have centralized data

Currently, human resources applications are distributed throughout HP. This makes it difficult to get various views of the organization. Different levels of management need to see different cuts of various pieces of human resources data and it has been costly, time consuming, and complex to merge data from various distributed databases together to get needed reports. In the past, divisions used to be completely contained in a certain geography. Currently, since some divisions are organized with parts spread throughout the country, even individual divisions are having difficulty keeping up with needed data on their businesses. But migration from the current distributed systems to a central system can't just automatically happen. Before data can be migrated to the central server from the current systems, data management must be done to standardize and/or correct the data.

PeopleSoft clients are set up throughout the country to connect to a central, U.S. Wide database server. If a major reorganization occurs (and it does frequently), then taking a different cut of the data should not be a problem. Different levels of management can get cuts of their data. In HP today, there are even managers, who are below the division general manager, who have geographically dispersed organizations.

An ideal solution for geographically dispersed organizations is to have a single, company-wide database system. But due to major differences in the way that international HP entities do business (different laws for each country; different needs for reporting; different languages), this initial system is a U.S. based system. HP's international businesses are working independently with PeopleSoft to develop human resources systems based on the way they do their business. They are using the same technology and platform that the PeopleBase program is using. After all of our work has been completed, it will be more clear as to what will be needed for a single, company-wide system.

Risks, Resolutions and Tips

In every decision, there are always issues and risks that come up. The following is a discussion of these risks, any resolutions, and any tips:

- Lawsuit

  In today's business environment, most successful companies are likely to be sued for one reason or another. PeopleSoft is no exception. At the time of HP's purchase, a lawsuit was pending against PeopleSoft.

  In order to lessen the risk, clauses were placed in HP's contract with PeopleSoft, which stated that HP had a certain amount of time to back out based on the lawsuit. Also, instead of PeopleSoft receiving money for all services, a certain amount of money due would be placed in escrow until the lawsuit was resolved or until a certain date passed, whichever came first.

Fortunately, the lawsuit was settled before the deadline, when HP had to decide whether to continue with the negotiated contract or not. The lifting of that cloud certainly helped HP feel more comfortable about the vendor-customer relationship, but working with successful small companies in the midst of potentially crippling lawsuits is a business mode that will probably be more prevalent in the future.

- Quality considerations and the issue of product control

  Fixing defects and producing enhancements to the product are in the control of the vendor. This can be a difficult position to be in compared to what HP was previously used to, and that is, developing and controlling all internal business applications.

  How this was overcome was that the human resources and payroll functional users accepted the product with what it did as is with the understanding that panels and records could be modified by tools (panel editor and record editor) provided by the vendor. What was also taken into account is that PeopleSoft has a certain way that they use Windows (can't do everything you can do with Windows in the panel editor) and has certain design standards that cannot be customized.

  In addition, a significant investment was made in developing a test system to be used for regression testing of this Windows product. Future releases of the vanilla PeopleSoft product and of HP's customized version will be tested using this system. It is highly automated and designed for minimal changes as ongoing customization occurs.

- Support considerations

  When a small company does support for a major human resources and payroll system for several Fortune 100/500 companies, there are always questions on how much support will they give in the future and on how long will they be around?

  As for how much support will they give in the future, it's hard to foresee for any small company. The best that could be done in this area is to pay for yearly support contracts, where payment/money could end at a specified time, if support was unsatisfactory. As with HP's computer business, support brings in a significant amount of revenue and is a mainstay of HP's survival. There was hope that this would be the same for PeopleSoft and that in order to get this revenue and ensure customer satisfaction, they would keep their support at a high level. In addition, there is a quarterly face to face meeting with the account management vice president and account managers servicing HP. This makes sure that the support relationship between PeopleSoft and HP is going well and that steps can be taken in order to correct problems.

Another potential help in support and technical expertise is outside consultants. As PeopleSoft's product has become more widespread, there have been a growing number of technical consultants, who work with companies to modify, implement, and support various aspects of the product.

- Vendor viability

  As with any small company vendor, there may be concern as to how long the company can survive. As seen in the mid to late 1980s, several software companies have had severe financial trouble and left their customers in great difficulty.

  Again, the strong tools that came with the product help mitigate the risk of vendor viability. If the vendor went away, product defect fixes and enhancements may not be able to occur, but panels, records, and menus could still be modified by HP.

- No track record with large numbers of users

  At the time of HP's purchase, PeopleSoft had just started to work with large Fortune 100 customers, but none of them had been in production. In addition, there were no large installations of the PeopleSoft product. At that time, a "large" DB2 operation had 8-10 PCs connected to an IBM mainframe and the entire operation was on one floor. HP needs to have at least several hundred connections and they would be spread across the U.S.

  As with any new development, there were no guarantees. To help mitigate the risk, HP Corporate enlisted the help of the HP 3000 and Allbase/SQL product division (Commercial Systems Division (CSY)). Corporate would gain the needed technical help and CSY would gain their first large scale customer reference on client/server Allbase.

- Added complexity of client/server

  Constructing a client/server environment is considerably more difficult than constructing a standard terminal based application. There are several more variables to worry about. If a problem occurs, the trouble could be caused by client hardware or software, network problems, or server hardware or software. The hope here is that by sticking to standards and by leveraging the support of standard configurations over multiple applications, there would be some economies of scale for troubleshooting problems. The PeopleBase program itself plans to support several client/server based applications U.S. wide and the information technology area plans to support others.

- Not invented here

  Going from home-grown tools to third party supplied software packages is a large jump for a development organization. There may be emotional considerations for development engineers not owning and controlling the software that they are using, but the method of project management is also quite different. There is significantly less control of the personnel and software involved. The schedule is much more in the hands of someone outside of the company and that can be difficult. In the past, some project managers had intimate technical knowledge of the tools and software. That detail knowledge is not always obtainable, if a third party is doing the product.

  Although the development team and management of the PeopleBase program can't obtain detailed level knowledge of how PeopleSoft was constructed and of how the panel editor/record editor/other tools were written, this group can develop the panels, put PeopleCode behind the panels, write server batch jobs, and write client and server reports. Once the development team completes more end user deliverables, it should feel more ownership for them, because that was invented here.

Choosing a Machine - HP 3000/9x7 running the MPE/ix Operating System

- PeopleSoft willing to port to HP 3000

  Initially, PeopleSoft's only two platforms were an IBM DB2 database on an IBM mainframe and a Gupta SQLBase database on an IBM PC compatible running OS2. Given the scope of the PeopleBase project and the amount of data involved, putting the corporate database on a PC was not feasible. Therefore, the only choice left was using an IBM mainframe as the database server. But, CSY was independently starting up a Value Added Business (VAB) relationship with PeopleSoft. Their strategy was to have a human resources and payroll software package on the HP 3000. In addition, there was a major added bonus to also have this application be an HP 3000 Allbase/SQL client/server application. For PeopleSoft, this was a way to have a mid-range product offering between DB2 and SQLBase. For the PeopleBase team, if a mid-range solution had enough horsepower, it was a major added plus to also have a solution that worked on HP equipment. Also, the cost of upkeeping a mainframe versus the internal transfer cost of upkeeping an HP 3000 was considered. With CSY's mainframe downsizing strategy, even if Corporate could not get internal pricing for HP equipment, using an HP 3000 was still a bargain over selecting a mainframe.

At that time, PeopleSoft didn't find it in their interest to connect to a Unix client or server so it was difficult to choose another server machine or another database platform.

- Help from CSY

  CSY was an eager partner in this relationship and was willing to tune Allbase/SQL, if there was real added value. They were looking for a showcase system and a good customer reference that would demonstrate that a third party, large, client/server application running on the HP 3000 with Allbase/SQL would work. Since then, they have provided significant help, especially in performance tuning. CSY has worked with the vendor to improve their performance, has spent a great deal of time and effort on improving network performance, has given significant technical support, and has lent equipment (HP 3000/967).

- Other marketed aspects of the HP 3000/9x7

  - Fast batch processing

    The HP 3000 is tuned to do batch processing well. This is especially important for payroll processing. Since the PeopleSoft product is mainly used for data entry, batch jobs are required to process the data.

  - Proven business machine

    For many years, the HP 3000 has serviced business applications well. Since PeopleSoft was seen as a business application, PeopleBase program management had more confidence in it.

  - Can handle large files well

    With large numbers of employee records being stored on the SQL database, there was always a concern as to how well the system would handle large files. This was something the HP 3000 has a proven track record on.

  - Fastest machine for Allbase

    Performance of the database engine was a crucial part of the decision on which database to use. HP's current payroll system on a mainframe computer has been optimized for performance over the last several years. One of the objectives of the PeopleBase program is to not give the users less than what they have now. But running payroll on

PeopleSoft using an SQL database has some performance hurdles to cross before it can approach the speed of the current payroll system.

So since Allbase is the fastest SQL database on both the HP 3000 and the HP 9000 platforms, it was an ideal choice.

- Migratability to HP-UX

  Since the HP 3000 and HP 9000 share the same hardware for a number of different product models, and because they also share many of the new product offerings, and because Allbase/SQL and C work the same way on both platforms, choosing the HP 3000 allows the flexibility to migrate to the HP 9000 server.

Project Management

All projects have their difficulties and the PeopleBase program is no exception. This paper will discuss some of the positive aspects on how the project was managed:

- Some customers on team

  This helped the project because:

  - Open communication lines

    This helped have direct communication lines with functional users. All too often, the functional human resources and payroll people have their own jobs to do and don't have time to spend to create or review detailed specifications. Having dedicated resources solves this problem. A major area to monitor is that functional users and development engineers have very different values and look at the world very differently.

  - Help avoid us and them

    Normally functional users somehow give specifications to a development team so that software can be developed. The development team then writes external specifications and gets feedback from the functional users on whether the specifications are satisfactory or not. After that, the development team concentrates solely on development work without functional user input. When the product is ready to be released, sometimes functional needs have changed and sometimes they don't get what they thought they would get.

Then begins the finger pointing. Having functional members on the team makes sure that they are informed and involved to some extent every step of the way.

- Senior management supportive of client/server

  It was very helpful that senior management felt that going toward a client/server environment and using more state of the art technology were the direction that business applications should go. For many years, COBOL/Image/Vplus was the technology of these applications and it was hard and would be hard to get out of that loop without the support of senior management.

- Buy-in from above through demos to senior managers

  The PeopleBase program is a multi-year program, but like all programs, it needs to be funded every year. In order to ensure that buy-in was in place before the product went too far, a skit demo of fairly wide ranging functionality was performed for HP's president and for other top managers over a three month period. The skit demos went very well and top management reaffirmed their buy-in to the program. If they had not, it was best to know then before significant effort and dollars were spent.

- Joint Product Evaluation Team (JPET)

  Monthly meetings with PeopleSoft's Allbase porting team were instrumental in checking the schedule, monitoring performance gains, and obtaining needed information. Questions and issues were raised on a wide ranging list of topics. In addition to these meetings, there were also phone conversations as needed with the vendor.

- Quarterly account management meetings

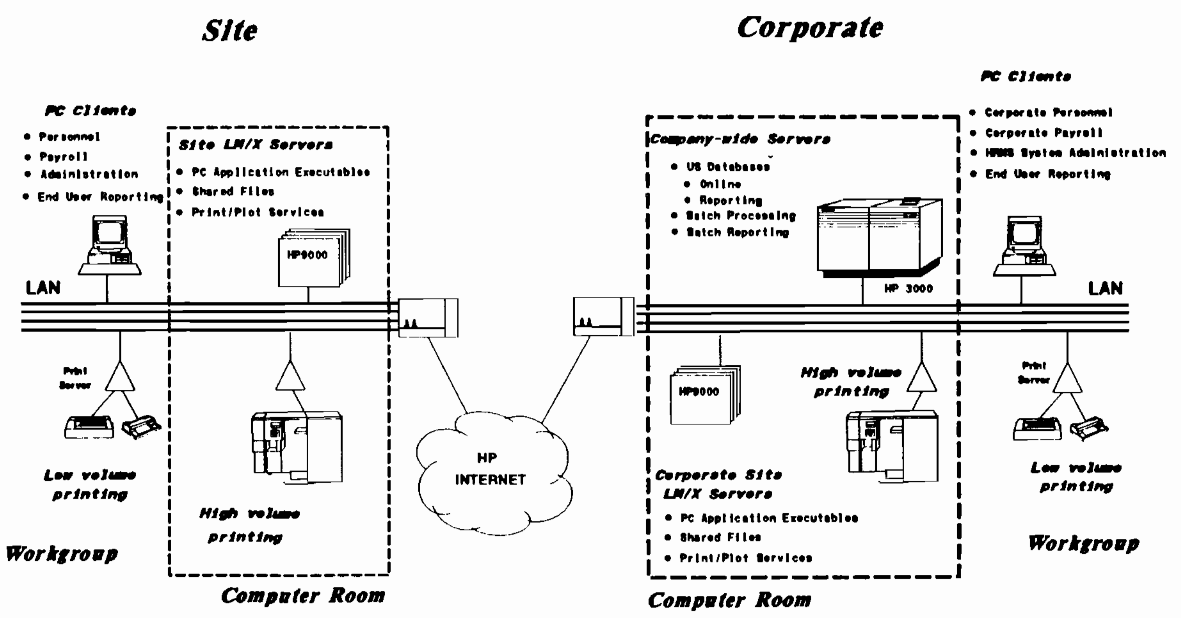
  The quarterly account management meetings were a vehicle to make sure that customer support was taken care of. This included on-line support, training, and other account management activities.

Appendix

The following illustrations give a graphic view of the Hewlett-Packard PeopleBase environment. Figure 1 shows the technical environment, where there is a central U.S. wide server at corporate (which could be replicated by Allbase/Replicate for more CPU power for batch processes and reporting) and remote sites. This is followed by Table 1, which shows the development and production hardware and software for the system. Then comes Figure 2, a sample menu and then comes Figure 3, a sample panel. The remaining figures go through the tools that are used to customize the vanilla PeopleSoft product so that it can be fitted for Hewlett-Packard's business environment. These tools are quite easy to use and are very rich in functionality. Classroom instruction on the use of these tools are offered by PeopleSoft.

To start the process of creating a working panel, a developer needs to use the record editor to create and maintain SQL tables. This is shown in Figure 4. Much can be done in the record editor including linking a field with an edit table or making a field a key or a list item. Figure 5 shows how PeopleCode can be placed behind a field to do processing, when a user uses a panel to do a row insert. If need be, PeopleCode can call a subroutine in a Windows DLL to execute code on the PC. Figure 6 shows the panel editor, where a panel can be created interactively by placing various widgets and by connecting them with fields in the database. After a panel is finished, it is placed on a menu. This is shown in Figure 7. Now the panel can be rigorously tested and released for general use. Figure 8 shows a very different type of tool called the Tree Editor. It allows a user to edit an organization tree and even link it to organizational security. Figure 9 is a tool which controls object security. A user's PeopleSoft login may only let him/her edit certain records, panels, menus, or perhaps other objects. And finally, Figure 10 shows SQLTalk for Windows, which allows a user to send SQL commands to the server and receive responses. Different users can have access to none, some, or all of these tools.

PeopleBase/HRMS Technical Environment



PeopleBase
HRMSTECH.GAL RKK 04/28/92

HEWLETT PACKARD

Figure 1
Implementing a Client/Server Application

Development (16 people + consultants)	Production (all hardware/software could be upgraded as new technology is released)
Client Hardware	Client Hardware
15 HP Vectra QS/20	HP Vectra QS/20 or better
4 HP Vectra 386/25	Prefer 486/33 for heavy use
3 HP Vectra 486/33	486/20, RS/25, 386/25 for med use
5 HP-UX Workstations	QS/20 for light use
All Vectras have	All Vectras should have
4 MByte Memory / Several have 8 Mbyte Memory	4 MBytes Memory (8 Mbytes may help performance)
LAN Wiring Board	LAN Wiring Board
VGA Color Display	VGA Color Display
Microsoft Mouse	Microsoft Mouse
Client Software	Client Software
MS DOS 5.0	MS DOS 5.0
Microsoft Windows 3.0	Microsoft Windows 3.0, 3.1
Lan Manager 1.2 on Ethernet	Lan Manager 1.2 on Ethernet
Arpa Services 2.1	Arpa Services 2.1
Network Services 2.1	Network Services 2.1
Allbase PC/API	Allbase PC/API
PeopleSoft 2.0	PeopleSoft 2.0+
SQR for Windows	SQR for Windows (for certain Corporate users)
C (create Windows DLL's)	
AutoTester for Windows	
NewWave Access	NewWave Access (ad hoc reporting)
	AmiPro (general text editor)
Lotus 1-2-3 for Windows	Lotus 1-2-3 for Windows (general spreadsheet)
MS Project (maintain schedule)	
Server Hardware	Server Hardware
HP 3000/957 Allbase Server	HP 3000/9x7 Allbase Server for updates
	HP 3000/9x7 Allbase Server for read-only reporting
HP 9000/852 LM/X Server	HP 9000/8xx LM/X Server
Server Software	Server Software
MPE-ix 3.1	MPE-ix 4.0
Allbase/SQL	Allbase/SQL
Allbase/NET	Allbase/NET
SQR (SQR runs on the server)	SQR (SQR runs on the server)
C Compiler (batch jobs/reports)	
Cobol Compiler (for conversion programs)	
xdb Debugger (debug C code)	

Table 1

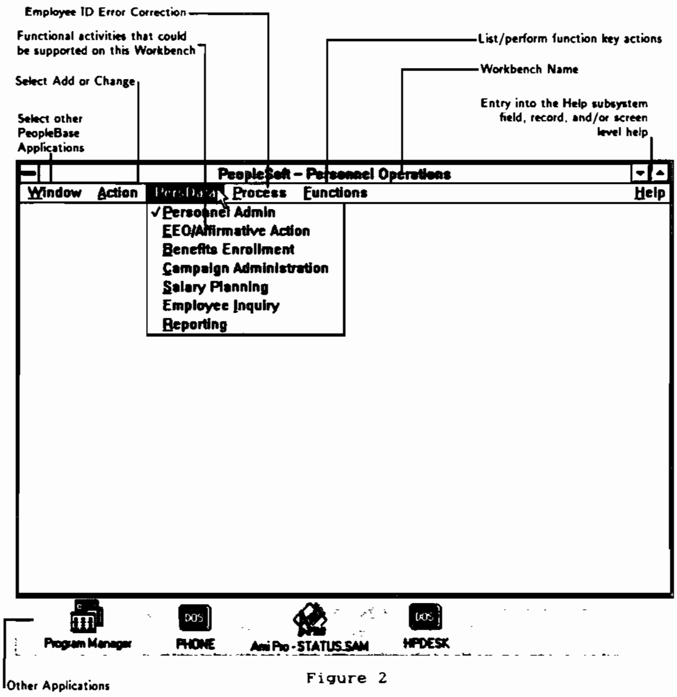
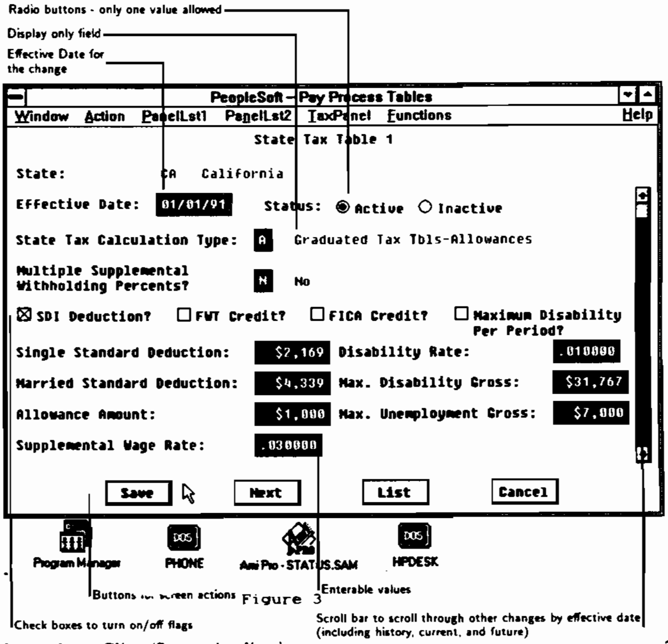

Employee ID Error Correction
Functional activities that could be supported on this Workbench
Select Add or Change
Select other PeopleBase Applications
List/perform function key actions
Workbench Name
Entry into the Help subsystem field, record, and/or screen level help
PeopleSoft – Personnel Operations
Window
Action
PersData
Process
Functions
Help
✓Personnel Admin
EEO/Affirmative Action
Benefits Enrollment
Campaign Administration
Salary Planning
Employee Inquiry
Reporting
Program Manager
PHONE
Ami Pro - STATUS.SAM
HPDESK
Other Applications

Figure 2


Radio buttons - only one value allowed
Display only field
Effective Date for the change
PeopleSoft – Pay Process Tables
Window
Action
PanelLst1
PanelLst2
TaxPanel
Functions
Help
Save
Next
List
Cancel
Program Manager
PHONE
Ami Pro - STATUS.SAM
HPDESK
Buttons for screen actions
Enterable values
Check boxes to turn on/off flags
Scroll bar to scroll through other changes by effective date (including history, current, and future)

State Tax Table 1

State: CA California

Effective Date: 01/01/91 Status: [✓ Active] Inactive

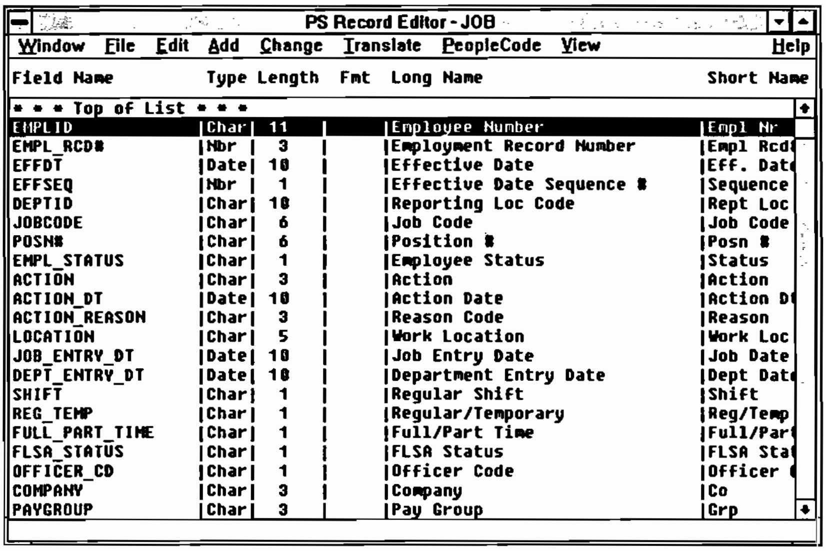
State Tax Calculation Type: A Graduated Tax Tbls-Allowances

Multiple Supplemental Withholding Percents? N No

☒ SDI Deduction? ☐ FWT Credit? ☐ FICA Credit? ☐ Maximum Disability Per Period?

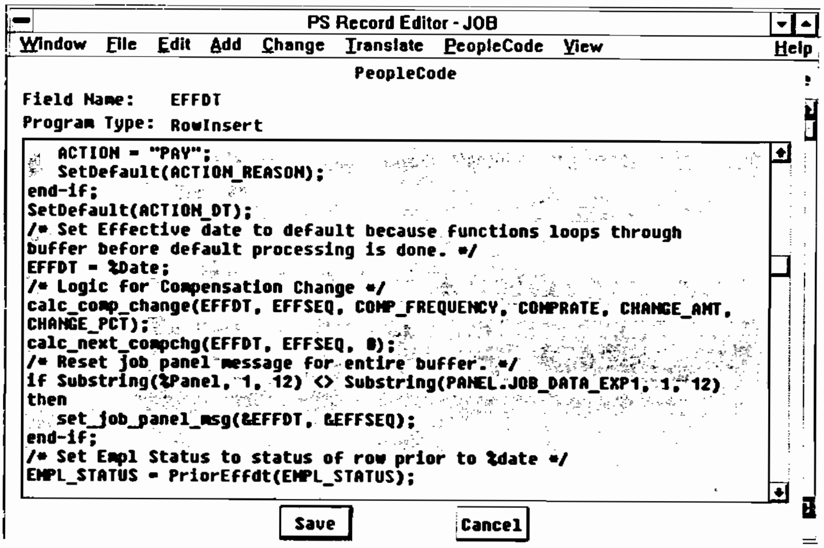
Single Standard Deduction:	$2,169	Disability Rate:	.010000
Married Standard Deduction:	$4,339	Max. Disability Gross:	$31,767
Allowance Amount:	$1,000	Max. Unemployment Gross:	$7,000
Supplemental Wage Rate:	.030000		

Figure 3



```
Field Name          Type  Length  Fmt   Long Name                       Short Name
* * * Top of List * * *
EMPLID              |Char| 11   |      |Employee Number                |Empl Nr
EMPL_RCD#           |Nbr |  3   |      |Employment Record Number       |Empl Rcd
EFFDT               |Date| 10   |      |Effective Date                 |Eff. Dat
EFFSEQ              |Nbr |  1   |      |Effective Date Sequence #      |Sequence
DEPTID              |Char| 10   |      |Reporting Loc Code             |Rept Loc
JOBCODE             |Char|  6   |      |Job Code                       |Job Code
POSN#               |Char|  6   |      |Position #                     |Posn #
EMPL_STATUS         |Char|  1   |      |Employee Status                |Status
ACTION              |Char|  3   |      |Action                         |Action
ACTION_DT           |Date| 10   |      |Action Date                    |Action D
ACTION_REASON       |Char|  3   |      |Reason Code                    |Reason
LOCATION            |Char|  5   |      |Work Location                  |Work Loc
JOB_ENTRY_DT        |Date| 10   |      |Job Entry Date                 |Job Date
DEPT_ENTRY_DT       |Date| 10   |      |Department Entry Date          |Dept Dat
SHIFT               |Char|  1   |      |Regular Shift                  |Shift
REG_TEMP            |Char|  1   |      |Regular/Temporary              |Reg/Temp
FULL_PART_TIME      |Char|  1   |      |Full/Part Time                 |Full/Par
FLSA_STATUS         |Char|  1   |      |FLSA Status                    |FLSA Sta
OFFICER_CD          |Char|  1   |      |Officer Code                   |Officer
COMPANY             |Char|  3   |      |Company                        |Co
PAYGROUP            |Char|  3   |      |Pay Group                      |Grp
```

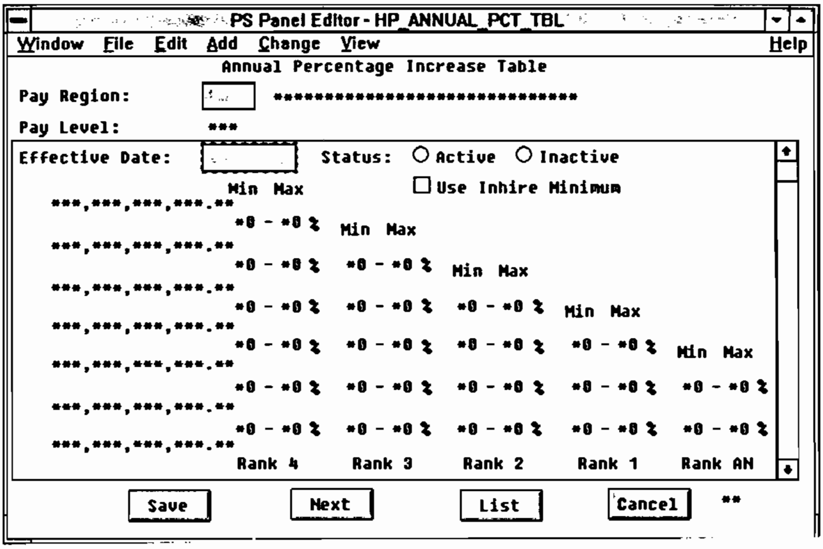
Figure 4



```
Field Name:   EFFDT
Program Type: RowInsert
```

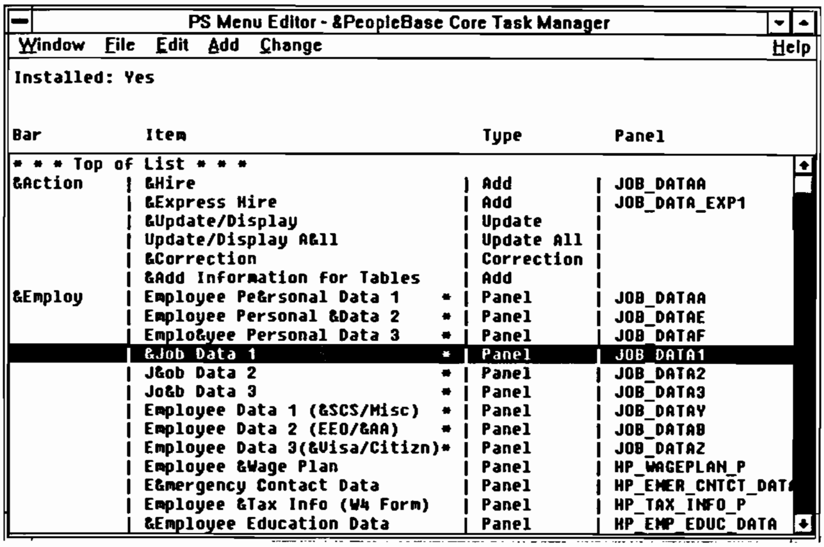
```
   ACTION = "PAY";
   SetDefault(ACTION_REASON);
end-if;
SetDefault(ACTION_DT);
/* Set Effective date to default because functions loops through
buffer before default processing is done. */
EFFDT = %Date;
/* Logic for Compensation Change */
calc_comp_change(EFFDT, EFFSEQ, COMP_FREQUENCY, COMPRATE, CHANGE_AMT,
CHANGE_PCT);
calc_next_compchg(EFFDT, EFFSEQ, 0);
/* Reset job panel message for entire buffer. */
if Substring(%Panel, 1, 12) <> Substring(PANEL.JOB_DATA_EXP1, 1, 12)
then
   set_job_panel_msg(&EFFDT, &EFFSEQ);
end-if;
/* Set Empl Status to status of row prior to %date */
EMPL_STATUS = PriorEffdt(EMPL_STATUS);
```

Figure 5



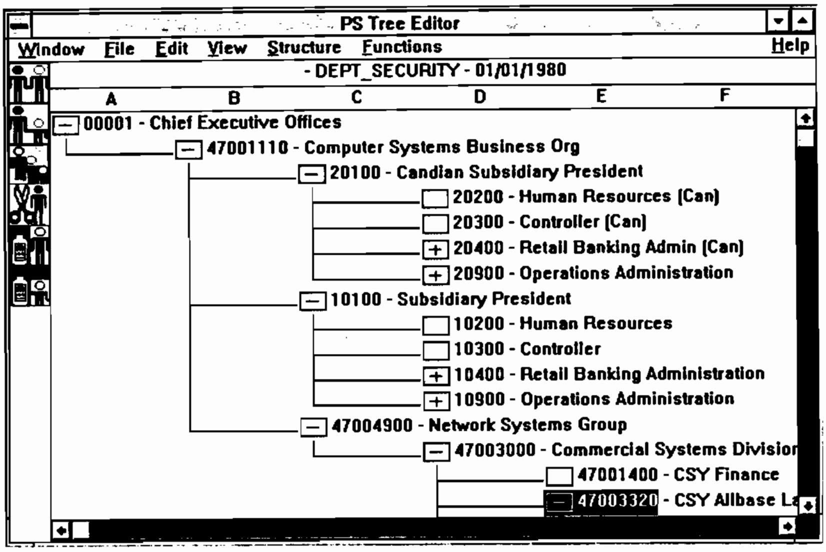
```
PS Panel Editor - HP_ANNUAL_PCT_TBL
Window  File  Edit  Add  Change  View                                   Help
                    Annual Percentage Increase Table
Pay Region:              ******************************
Pay Level:               ***
Effective Date:                Status:  O Active   O Inactive
                                        ☐ Use Inhire Minimum
                       Min  Max
  ***,***,***,***.**
                        *0 - *0 %  Min  Max
  ***,***,***,***.**
                        *0 - *0 %  *0 - *0 %  Min  Max
  ***,***,***,***.**
                        *0 - *0 %  *0 - *0 %  *0 - *0 %  Min  Max
  ***,***,***,***.**
                        *0 - *0 %  *0 - *0 %  *0 - *0 %  *0 - *0 %  Min  Max
  ***,***,***,***.**
                        *0 - *0 %  *0 - *0 %  *0 - *0 %  *0 - *0 %  *0 - *0 %
  ***,***,***,***.**
                        *0 - *0 %  *0 - *0 %  *0 - *0 %  *0 - *0 %  *0 - *0 %
  ***,***,***,***.**
                        Rank 4     Rank 3     Rank 2     Rank 1     Rank AN

           Save          Next          List          Cancel    **
```

Figure 6



```
PS Menu Editor - &PeopleBase Core Task Manager
Window  File  Edit  Add  Change                                         Help
Installed: Yes

Bar         Item                              Type          Panel
* * * Top of List * * *
&Action   | &Hire                           | Add         | JOB_DATAA
          | &Express Hire                   | Add         | JOB_DATA_EXP1
          | &Update/Display                 | Update      |
          | Update/Display A&ll             | Update All  |
          | &Correction                     | Correction  |
          | &Add Information for Tables     | Add         |
&Employ   | Employee Pe&rsonal Data 1     * | Panel       | JOB_DATAA
          | Employee Personal &Data 2     * | Panel       | JOB_DATAE
          | Emplo&yee Personal Data 3     * | Panel       | JOB_DATAF
          | &Job Data 1                   * | Panel       | JOB_DATA1
          | J&ob Data 2                   * | Panel       | JOB_DATA2
          | Jo&b Data 3                   * | Panel       | JOB_DATA3
          | Employee Data 1 (&SCS/Misc)   * | Panel       | JOB_DATAY
          | Employee Data 2 (EEO/&AA)     * | Panel       | JOB_DATAB
          | Employee Data 3(&Visa/Citizn)*  | Panel       | JOB_DATAZ
          | Employee &Wage Plan             | Panel       | HP_WAGEPLAN_P
          | E&mergency Contact Data         | Panel       | HP_EMER_CNTCT_DAT
          | Employee &Tax Info (W4 Form)    | Panel       | HP_TAX_INFO_P
          | &Employee Education Data        | Panel       | HP_EMP_EDUC_DATA
```

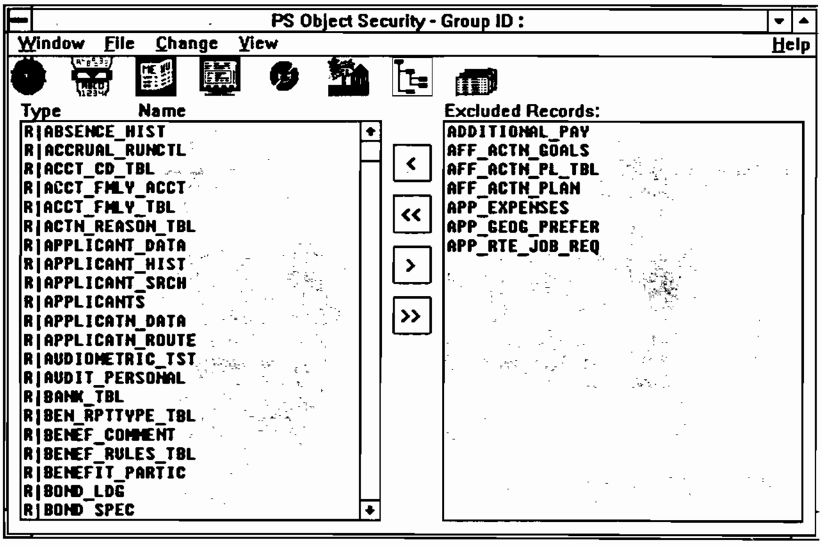
Figure 7



- DEPT_SECURITY - 01/01/1980

```
[-] 00001 - Chief Executive Offices
    [-] 47001110 - Computer Systems Business Org
        [-] 20100 - Candian Subsidiary President
            [ ] 20200 - Human Resources (Can)
            [ ] 20300 - Controller (Can)
            [+] 20400 - Retail Banking Admin (Can)
            [+] 20900 - Operations Administration
        [-] 10100 - Subsidiary President
            [ ] 10200 - Human Resources
            [ ] 10300 - Controller
            [+] 10400 - Retail Banking Administration
            [+] 10900 - Operations Administration
        [-] 47004900 - Network Systems Group
            [-] 47003000 - Commercial Systems Divisio
                [ ] 47001400 - CSY Finance
                [-] 47003320 - CSY Allbase La
```

Figure 8

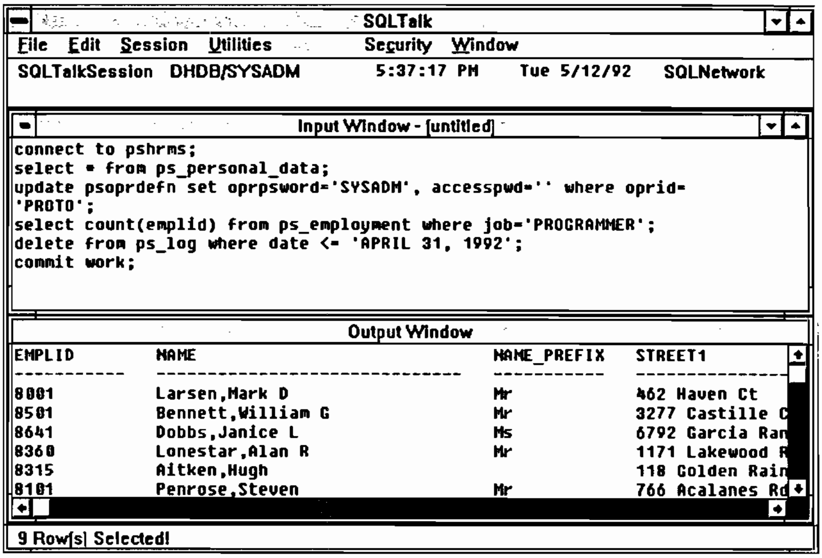


Type	Name
R	ABSENCE_HIST
R	ACCRUAL_RUNCTL
R	ACCT_CD_TBL
R	ACCT_FMLY_ACCT
R	ACCT_FMLY_TBL
R	ACTN_REASON_TBL
R	APPLICANT_DATA
R	APPLICANT_HIST
R	APPLICANT_SRCH
R	APPLICANTS
R	APPLICATN_DATA
R	APPLICATN_ROUTE
R	AUDIOMETRIC_TST
R	AUDIT_PERSONAL
R	BANK_TBL
R	BEN_RPTTYPE_TBL
R	BENEF_COMMENT
R	BENEF_RULES_TBL
R	BENEFIT_PARTIC
R	BOND_LOG
R	BOND_SPEC

Excluded Records:

```
ADDITIONAL_PAY
AFF_ACTN_GOALS
AFF_ACTN_PL_TBL
AFF_ACTN_PLAN
APP_EXPENSES
APP_GEOG_PREFER
APP_RTE_JOB_REQ
```

Figure 9


SQLTalk
File Edit Session Utilities Security Window
SQLTalkSession DHDB/SYSADM 5:37:17 PM Tue 5/12/92 SQLNetwork
Input Window - [untitled]
Output Window
9 Row[s] Selected!

```
connect to pshrms;
select * from ps_personal_data;
update psoprdefn set oprpsword='SYSADM', accesspwd='' where oprid=
'PROTO';
select count(emplid) from ps_employment where job='PROGRAMMER';
delete from ps_log where date <= 'APRIL 31, 1992';
commit work;
```

```
EMPLID       NAME                            NAME_PREFIX   STREET1
-----------  ------------------------------  -----------   ---------------
8001         Larsen,Mark D                   Mr            462 Haven Ct
8501         Bennett,William G               Mr            3277 Castille C
8641         Dobbs,Janice L                  Ms            6792 Garcia Ran
8360         Lonestar,Alan R                 Mr            1171 Lakewood R
8315         Aitken,Hugh                                   118 Golden Rain
8101         Penrose,Steven                  Mr            766 Acalanes Rd
```

Figure 10

HP 3000 Multiprocessing: Synchronization & its Performance Implications

Paper #3858

Kim Rogers, Marsha Duro, Craig Hada, Steve Saunders

Hewlett-Packard Company
19447 Pruneridge Avenue
Cupertino, California, USA, 95014
(408) 725-8900

Abstract

Hewlett-Packard Company is offering HP 3000 customers greater capacity and performance through the technology of multiprocessing. This paper will present an introduction to the software issues of multiprocessing with an emphasis on the concepts of synchronization. The performance implications of multiprocessing and the variability and dependence of MP performance on workload characteristics will be discussed. Examples will be drawn from MP performance tuning experiences on multiprocessor HP 3000 systems using internal on-line transaction processing (OLTP) and batch benchmarks.

Introduction

In 1990, Hewlett-Packard Company introduced its first multiprocessor computer system based on its own PA-RISC architecture, the HP 3000 Series 980/200. Multiprocessing has allowed customers to add processing power seamlessly to systems to relieve CPU bottlenecks in a manner identical to adding memory to relieve memory bottlenecks. Since then, Hewlett-Packard Company continues to offer HP 3000 customers high-end performance growth through faster processors and improved multiprocessing (MP) technology. This is particularly true with the recent introduction of the HP 3000 Series 992.

On most HP 3000 systems, the availability of many users accessing the system provides the opportunity for greater performance with additional CPUs. However, the effective increase in CPU capacity due to multiprocessing is not strictly linear in the number of processors. In other words, adding a second CPU to a system rarely doubles the system performance. A workload with a high degree of multiprogramming and independence of tasks (applications) will exhibit better performance, in a multiprocessing environment.

This paper will discuss some important aspects of MP systems and how they influence system performance. Examples from actual performance analysis from early MPE/iX MP prototype systems will be presented. These examples, although drawn from MPE/iX experiences, illustrate performance implications of multiprocessing systems. These principles apply to all activity that may execute on a computer system.

Concepts of Synchronization

Like all operating systems, MPE/iX is primarily a resource manager. Its fundamental function is to manage and schedule the system resources to maximize the amount of useful work. The hardware resources include CPUs, main memory, secondary storage (disks), along with data communication and other input/output devices. Single users rarely fully use all these resources simultaneously. For efficiency, the operating system allows multiple users access to the system and manages the service requirements of these users to utilize the hardware optimally. Thus, while one process is executing in the CPU, others may be waiting for service by the disks or input from terminals. The operating system continually monitors the state of all processes and their service requirements and allocates resources accordingly.

In a multi-user computer system, software resources are another critical class of resources that must be managed. As anyone with a joint checking account knows, access (check writing) against an account must be coordinated to avoid loss of integrity (bounced checks). In a computer system, access to data must be synchronized. Locks are the general synchronization mechanism used to control access to shared resources. This data may be user data, such as database files; or system data, such as internal tables used to manage CPU, memory, or processes. A TurboImage application may coordinate access to changeable data by performing a DBLock/DBUnlock around a "transaction" (consisting of one or more calls to DBGet and DBUpdate). By locking the data, process "A" ensures that process "B" may not access the data until process "A" unlocks the data.

Locks can be defined to control individual records, whole files or data bases, or whole applications. When a process requests a lock already owned by another process, it must wait for the lock to become available. This situation, called contention, can have a negative impact on system performance.

Lock Contention

Usually, lock contention is handled by stopping the execution of the requesting process until the lock is released. This action is called a process stop (or process block). The essential point is that a process, unable to get a lock, gives up the CPU. This is handled entirely within the lock primitive. The lock primitives will block requestors if the lock is currently owned by another process. Since code is executed sequentially, it is not possible to perform further work without successfully obtaining the lock. By blocking (the requestor), the owner or another process can be given access to the CPU resource to complete its task. When the owner releases the lock, the requestor will be unblocked and be given ownership of the lock. When the requestor returns from the lock primitive, the requestor will be the new owner of the lock. The requestor of the lock cannot tell whether it contended for the lock, or was granted the lock immediately.

The actual implementation of locks internally within MPE/iX takes two forms. Understanding them is useful in understanding the issues of multiprocessing. The internal lock forms are:

- Block the process upon contention (normal locks), and
- Spin-wait upon contention (spin locks).

The vast majority of locks are "normal". Examples of normal locks externally available from MPE/iX are database (DBLock) and file (Flock) locks. Process blocks have the advantage of releasing the CPU (a critical system resource) to another task. Process blocks, due to lock contention, are a source of overhead that can diminish system throughput. Low level areas of the operating system cannot block upon contention and must therefore spin-wait until the lock becomes free. These "spin" locks are only useful within the operating system, but understanding them is useful in understanding multiprocessing issues. Spin locks will "burn" CPU cycles that will not be available for any other task. Naturally, spin locks are used sparingly and only for internal operating system locks having very short hold times.

Throughput and Response Time

Management of system resources, both hardware and software, is transparent outside the operating system, but is not free. The more resources that the operating system manages, the greater the overhead. Further, just as performance can be degraded by a hardware resource bottleneck, software resource contention similarly impacts performance. This is often measured by the most common metrics of performance analysis, throughput and response time. Response time is a measure of how long an individual user must wait for a response from the system. It is merely the sum of all the service and wait times. Throughput is a measure of how many responses the system makes to the entire set of users in a given unit of time. Both hardware and software resource contention results in decreased throughput and increased response times.

A trade-off that occurs in systems is that maximum throughput may come at the expense of degrading some or all user's response times. Minimum thresholds for response times are usually defined for a given environment (or benchmark). Therefore, while it may be possible to achieve a higher level of overall throughput on a system, one cannot totally ignore the impact to response times. This balance of the throughput and response time is typically defined by the system manager. By monitoring these components of performance, it's possible to determine if (or when) an upgrade to the system is needed to increase performance.

Multiprocessor Overview

The multiprocessor organization of PA-RISC architecture provides a tightly-coupled organization (see Figure 1). The processors reside on a common bus and access shared memory. Since I/O devices are memory mapped, each processor may access the I/O devices directly. Thus, all processors have equal access to both memory and I/O devices.

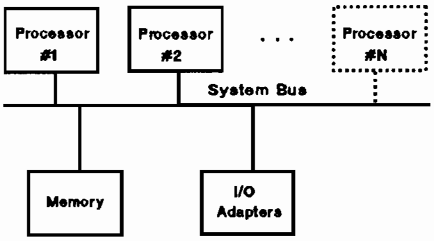


Figure 1. PA-RISC Multiprocessor Organization

MPE/iX supports this tightly-coupled, shared memory multiprocessor configuration and is designed to be a symmetric[1] multiprocessing operating system. This means that there are no inherent restrictions regarding which processor a specific activity may execute in. This approach provides the maximum opportunity to utilize the additional CPU capacity. MPE/iX exploits parallelism at the process level by scheduling processes to run concurrently. Thus, system performance is dependent upon the degree of multiprogramming exhibited by the workload (applications) and the degree of concurrency supported by the operating system and applications. Besides improving system performance, the MP system is completely transparent to users and is fully compatible with existing application code.

From the user's perspective, the multiprocessing environment provided by MPE/iX is the same as the MPE environment has been for many years. A multiprocessor HP 3000 is simply a uniprocessor HP 3000 with additional CPUs (Figure 1). These systems are called 2-way, 3-way, and 4-way multiprocessors. Currently, the HP 3000 Series 980 and Series 992 systems support MP capabilities. Additional CPUs make it possible to

[1] With asymmetry, a specific CPU is defined to handle a specific operating system function -- even if that function were needed by a process executing in another processors. This can often result in system bottlenecks.

execute more than one process simultaneously, the apparent simultaneousness of multiprogramming now becomes reality (with the additional CPUs).

MPE/iX still maintains a single, priority ordered, dispatcher queue for all processes ready to use CPU resources. The CPUs, and the processes they execute, share the main memory and the I/O devices of the system. All shared elements require synchronization to ensure the consistency of those resources. For example, in a 2-way multiprocessor both CPUs can use the dispatcher simultaneously but each must pick a different process to run or chaos will result. Chaos is avoided by requiring the dispatcher, on each CPU, to lock the dispatcher queue prior to selecting a process to execute. (The type of lock required for the dispatcher queue is an example of a spin lock.)

This can be illustrated by Figure 2. A single line of customers is serviced by multiple servers (bank tellers). Clearly, it does not make sense to have more bank tellers than the total number of people receiving and ready for service. If some people were to step out of line, then some tellers could be idle. If there is only one customer (one task in the system), opening up another teller window will not enable that single customer to receive faster service. If there is a long line of customers, and only a single bank teller, then opening another teller window enables the bank to provide better overall service. The amount of time a teller takes to perform a customer's transaction depends upon the type of transaction, not how many tellers are present. Therefore, the time that the customer actually spends with the teller will not be made faster with additional tellers present. However, the line moves faster since two (or more) customers are serviced simultaneously. This is exactly the idea behind multiprocessors. Adding more processors to a system is beneficial when there are multiple tasks that can be executed (independently) on the processors simultaneously.

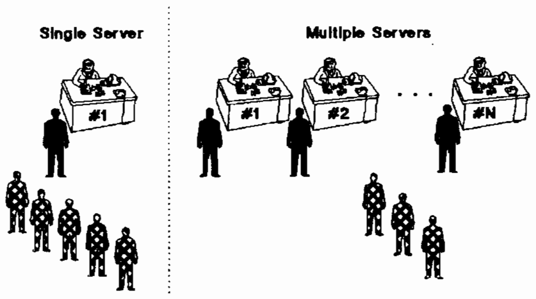


Figure 2. Concurrent Application Execution

Synchronization Applied to MPE/iX

Software, and its implementation of synchronization, can be viewed as multiple layers built on top of each other. The application layer sits above the operating system layer and requests services from the operating system to perform its tasks. The application layer cannot make assumptions about the progress or execution state of the other application processes. Cooperation among these processes must be controlled through explicit synchronization (e.g, RINs, file locks, message file IPC). Many operating system services are executed for the user process. These layers of the operating system are reentrant, there may be multiple invocations of these services that are active (within multiple processes). High level layers of the operating system protect the integrity of its data structures from multiple invocations via normal locks. The low level (kernel) makes use of spin locks.

All forms of locks rely upon an atomic operation (provided by PA-RISC) to guarantee exclusive access (across all processors) to the actual lock. All forms of MPE/iX locks rely upon this basic operation. Thus, the synchronization techniques, available to all levels of the system, work correctly in multiprocessor configurations. However, the probability for contention increases due to simultaneous activity on all CPUs. In a uniprocessor, lock contention occurs only if the holder of the lock loses the CPU before releasing the lock. The probability of contention is a function of the lock hold time. In a multiprocessor configuration, the possibility exists that multiple processes on multiple CPUs will require the same lock simultaneously. This results in increased lock contention.

Internal Benchmark Descriptions

The relevance of these concepts to system performance was demonstrated repeatedly during the development and analysis phase of the multiprocessing project. Internal benchmarks were used to measure and tune system performance. These benchmarks represented controlled, reproducible, and scalable workloads. Extensive instrumentation was used to measure different internal system performance parameters. Ultimately, system performance was characterized in terms of maximum throughput and utilization.

Three workloads were used most extensively. The first was an interactive banking application. This workload was very homogeneous. Each transaction represents a simple, update intensive database transaction requiring a single terminal read and write. The workload spends very little time within the application code. Over 90% of the time is spent within the database and operating system, less than 10% of the time is within the application.

The second workload was an interactive, manufacturing database application. This workload has three separate components. Each component contains multiple transactions of differing complexity. The terminal I/O component is also more complex, containing both character and block mode transmissions.

The last workload extensively used was an MRP batch workload. Multiple copies of this workload were created to increase the load. Separate, unique databases were used for each copy of this workload.

MP Performance Tuning: Experiences with Internal Benchmarks

Internal instrumentation was used extensively to analyze system performance. Additional internal instrumentation was created to identify the locks and measure lock frequencies used by the system under the various workloads. This instrumentation provided the stimulus for the adjustments of many algorithms. This effort resulted in a reduction in the number of lock/unlock operations and a significant improvement in the system performance across all benchmarks.

Prior to the MP release, the operating system kernel made extensive use of a single lock to control simultaneous accesses to a variety of resources. This form of locking, called "coarse grained", has the effect of reducing concurrency since independent activity is controlled by a single lock resource. This creates "artificial" contention and would have been inefficient in a multiprocessor configuration by restricting concurrency -- thus reducing overall system throughput.

The two major techniques used to control coarse grained locking are:

- Minimize the lock hold times. Release the lock periodically to create opportunities for other processes to acquire the lock.

- Identify independent tasks or data that is protected by the lock. Create separate locks to protect these independent items.

The MP prototype operating system was first modified to minimize the locks hold times. The lock was held for the shortest time possible before releasing the lock -- even if that meant repeatedly locking and unlocking the same lock. This sought to allow another process the opportunity to acquire the lock without encountering contention. This technique increased path lengths and resulted in reduced throughput for all workloads. The larger the percentage of total path spent in the operating system, the greater the impact to performance -- although all workloads were impacted negatively.

One must balance the number of lock and unlock operations with the actual level of concurrency created. If the additional "windows" of lock availability do not contribute to greater concurrency, the overall throughput of the system will not improve. In fact, throughput may actually diminish since the cost of the lock operations will not be counter balanced with any other gain. Therefore, the number of lock and unlock operations (that did not create significant gains in concurrency) were reduced so that the locking overhead would be reduced. This reduction in overhead improved performance.

To address the independence of tasks, more locks were defined to control smaller amounts and independent data. This does not imply that more lock and unlock operations are employed. Consider the example where lock "A" was originally designed to access data "x" or "y". If process 1 needs to access data "x", it first acquires lock "A". If process 2 needs to access data "y", it also needs lock "A". Thus, process 2 could be blocked waiting for lock "A" although it wants to access a separate data structure. Lock "A" can be replaced with locks "Ax" and "Ay", which are defined to protect accesses to data "x" and "y". Process 1 and process 2 will no longer be forced into contention artificially. Each process still performs a lock prior to their data access. Now each lock is defined to control a separate piece of data.

As described earlier, long lock hold times increase the likelihood for contention. With spin locks, contention begets spinning -- wasting CPU cycles. With blocking locks, contention produces additional process blocks. When a process holding a lock is blocked, possibly for a disk I/O or another lock, the lock hold time may become large. While the lock owner is blocked, many other processes may request the lock. This, in turn, causes those processes to block before the owner can relinquish the lock. As processor speed increases, and levels of concurrency increase, this scenario becomes more probable. Increased process stops are a large source of system overhead resulting in reduced throughput and increased response times. By analyzing the activity performed while the lock is held, it was possible to move activity outside the lock. In some cases, it was possible to avoid known activities that would lead to a process block while the lock was held. Thus, lock hold times were reduced. Again, by reducing the hold times, the levels of contention were diminished resulting in improved system throughput.

The most frequently used blocking locks within MPE/iX had short hold times. Analysis of the early MP system showed that processes were still blocking on the heavily used ("hot") operating system locks. In an MP environment, it's possible for two processes to request the same lock at effectively the same time. In a homogeneous workload, where all users are requesting the same services, if a lock is used heavily, contention will occur -- even with short hold times.

Consider the following: two processes execute exactly the same code, on different processors, and request the same lock. If available, the first to request the lock will be granted ownership immediately. If the first process releases the lock in fewer cycles than it takes to block and relaunch the second process, then it is most efficient for the operating system to put the second process into a busy-wait state until the lock is released. Otherwise, it is more efficient to block the second process.

To address this, a new lock algorithm was created. In a multiprocessor environment, when a process requests an unavailable lock, the operating system first checks to see if another process is already blocked for this lock. If so, the requesting process is blocked immediately. If not, the requesting process is put into a busy wait state for a predetermined time. During this time, if the lock is released, the process stops spinning, acquires the lock and continues processing. If the lock is not released within this short time, the process will be blocked. The high frequency locks with short hold times within the operating system were converted to this lock algorithm. MP performance improved significantly.

The performance improvements resulting from all these tuning efforts varied by workload. Naturally, the amount of improvement was dependent on the degree to which the workload was impacted by these problems. The banking application, with its simple homogeneous workload, places a high demand on identical operating system services at effectively the same time. In other words, it has a lower amount of independent tasks. Therefore, it achieves the lowest MP performance scaling. The manufacturing application has a variety of transaction types and accesses a broader cross section of operating system service requirements. This moderate level of independent tasks enables it to more fully exploit the added capacity of MP. With multiple instances of the MRP batch workload, behaving as distinctly independent workloads, there is little need for synchronization. As a result, the MRP workload exhibits the strongest MP scaling factor.

Conclusions

MPE/iX, like any computer system supporting multiprogramming, needs synchronization to control access to shared data. The low level synchronization embedded in the kernel is crucial to MP performance, but transparent to the user. There is no need for application software to be aware of the existence of multiple processors. In MPE/iX, applications are generally not aware of anything other than processes. Multiprocessing does not affect database, file, or application synchronization.

In most systems, the availability of many users logged on to the system provides the opportunity for greater performance via multiprocessing technology. To maximize that performance gain, it is important to have independent processes so that simultaneous CPU activity may occur. Adding CPU capacity will help those systems experiencing a CPU bottleneck. However, a system that is limited by I/O bottlenecks will generally not improve by adding CPUs.

If an application is divided into multiple processes, to achieve more independence, there are various intrinsic services available to the application to provide proper synchronization and inter-process communication. These basic intrinsic services allow applications to be divided into smaller components that may execute as separate processes -- while still sharing data and synchronizing activities. The key factor in predicting the performance of applications is the level of concurrency supported. Performance of applications in a multiprocessor environment is greatly influenced by the synchronization used:

- The addition of lock/unlock operations can add significant overhead. One must trade-off the potential concurrency of multiple activities with the cost of the locks to ensure data consistency.

- Excessive contention due to long lock hold times can be problematic. Unnecessary tasks should be moved outside the lock to ensure the shortest possible lock hold times. Other resources needed while a lock is held, that may result in process blocks, should be analyzed to reduce the probability of blocking.

- Coarse-grained locks should be avoided. Such locking schemes reduce the levels of concurrency. Thus, the performance potential of the application or system can be diminished.

The essential lesson is that synchronization has the most significant impact to MP performance. Increasing the granularity of locks (and reducing the overhead of locks), creates the independence necessary to utilize the additional CPU capacity offered by multiprocessors completely.

SOFTWARE RESILIENCY
3859

by

Kendall Sutton and Jessy Hsu
19447 Pruneridge Ave.
Cupertino, CA 95014
408-447-5645, 408-447-5676

Abstract

Operating system software failures are a major cause of unexpected downtime, particularly on larger systems running a more heterogeneous and stressful workload. System software also tends to react very poorly to hardware anomalies that occur. With the performance and connectivity growth of HP 3000 systems leading us rapidly into the mainframe class environment, the resiliency of the OS software will become a more critical factor in the overall availability of the system. As part of the MPE/iX high availability strategy, software resiliency is a focal point of attention. This paper examines some of the reasons that systems fail, discusses some of the strategies being employed and some specific implementations that have taken place to increase the resiliency of MPE/iX system software.

Introduction

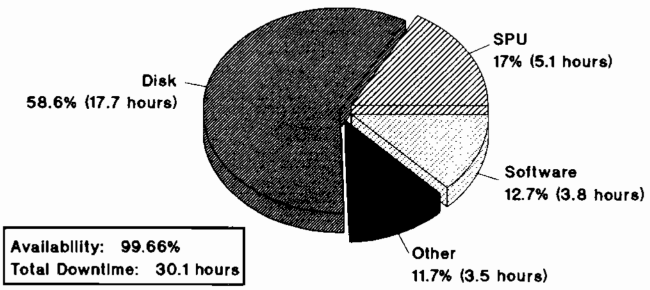
HP 3000 systems consistently rank at the top of their class in the area of system reliability. In order to meet ever increasing needs in the area of system availability, HP is taking pro-active steps to further improve the reliability of MPE/iX systems. The current focus on software resiliency related features represents the next logical step in the development of the HP 3000 high availability architecture.

HP 3000 High Availability Strategy

For the past several years, HP 3000 systems have followed a course of increasing system reliability. With the introduction of PA-RISC (Precision Architecture--Reduced Instruction Set Computing), HP redefined quality commercial systems. Based on early warranty data, the HP 3000 achieved well over 55% increase in reliability over the industry-acknowledged highly reliable Series 70. PA-RISC has been a significant factor in a leading industry watcher, Datapro, annually rating HP's reliability superior to all others, including IBM and DEC.

More recently, the objective has been to provide an integrated high availability architecture. It is built on a solid foundation of high availability features designed into PA-RISC based systems. These features range from exceptionally reliable hardware, to Predictive Support/iX which can predict maintenance needs before a failure can happen, to HP PowerFail, which provides uninterrupted battery backup after a power failure has occurred.

The high availability architecture focuses on minimizing or eliminating fundamental causes of both planned and unplanned downtime. This paper focuses on unplanned downtime. The strategy has involved addressing the major causes of unplanned downtime in order of their importance.



Source: HP 1990

Unplanned downtime due to hardware failures is addressed today by products such as Mirrored Disk/iX and the recently released PDA (Parity Disk Array) product, which address the disk failure issue and offer multiple levels of protection at a varying cost structure. SPU Switchover/iX allows for rapid recovery in the event of an SPU hardware failure. The next step in the architecture is to extend such concepts as predictive support and self managing systems into the system software arena in a way that further increases data availability on the HP 3000.

Resilient Software

The overall goal of the software resiliency project is to provide precision system software which is resilient to software and hardware exception conditions. Precision software is developed using a set of standard error handling methodologies and practices that apply to both the internals of a module as well as to module interactions. It is software that pro-actively takes steps to avoid problems before they occur and notifies operations of potential problems while there is still time to correct them. Exception conditions are errors or unexpected conditions that impede normal processing. Resilient means that when exception conditions are encountered the system is able to recover or gracefully degrade in performance or function.

Medical Model

In developing an overall software resiliency strategy, it is useful to have a model to assist in understanding how the various individual tactics fit together to form a consistent whole. A medical analogy provides such a framework.

An exception condition that arises in a computer system can be viewed as similar to an illness that occurs in human beings. There are a number of activities associated with human illness:

o Prevention

-- Actions can be taken to decrease the likelihood that an illness will occur or to lessen the severity if it does occur.

o Detection/Diagnosis

-- This must be performed in order to understand the nature of the particular ailment.

o Quarantine

-- This may be necessary in order to contain the disease and prevent it from spreading throughout the community.

o Treatment/Recovery

-- Obviously necessary for continued productivity.

All of these activities have direct counterparts in software resiliency. Preventing the occurrence of exception conditions, whether through rigorous design and coding methodology or self managing system functions, is a key factor. When an exception condition does happen, it's vital that the effects of it are contained and no corruption is allowed to spread to user data and applications. Proper diagnostic tools are necessary to understand the nature of the exception condition so that its cause can be determined and corrected.

Today, many exception conditions lead to a system abort. The challenge is to understand how system aborts can be eliminated while still maintaining high standards of data integrity.

System Abort Elimination

While much of the following discussion will deal with system aborts and system abort strategy, the concept of software resiliency involves more than that. When a system is running, situations will occur where operations need to be aborted. Along with minimizing the probability of a system abort, the overall direction of resiliency strategy includes decreasing the likelihood that an operation will need to be aborted and reducing or eliminating the impact to business applications when the abort of an operation is necessary. A generalized abort strategy involves virtually all aspects of error handling and recovery and therefore provides the foundation for software resiliency.

It is also important to understand the vital functions that many system aborts provide today.

Functions of System Abort

It may be surprising to note that three of the vital functions that system aborts provide map directly into the medical model presented earlier.

- Containment - When a situation arises that could lead to corruption, it is necessary to fail fast, before the contamination can spread.

- Diagnosis - Because the system abort allowed the system to fail fast, the DUMP facility is able to capture the state at that instant. This greatly facilitates the analysis of the problem.

- Recovery - It may seem unusual to think of a system abort providing recovery from a problem, but in fact, by rebooting after the abort, the system is usually able to continue normal operation. It has been said that most error conditions are transient and that simply retrying a transaction will lead to a successful completion about seventy-five percent of the time. Unfortunately, retrying the transaction means going all the way back to the generation of the system itself by rebooting.

Not all system aborts fit precisely within this model. For example, aborts that occur because of scarce resources are not used for containment purposes. These will be addressed by employing a self-managing system function that will take automatic action on the system when critical resources reach dangerously low levels. This will be discussed further in a later section.

High Level Resiliency Strategy

There are system aborts that exist within the operating system that provide these vital functions and there are some that do not. The strategy is to understand into which category the system aborts that are being encountered fall.

For those that provide a vital function, alternatives to aborting the system are being developed that can provide those same functions. Other system aborts that don't provide a critical functions are being investigated to see if they can be eliminated through other ways, such as adherence to error handling standards.

Two of the main goals of the software resiliency project are to reduce the frequency of system aborts and to lessen the impact to the user when an operation needs to be aborted. In order to do this we are developing an abort strategy that will fulfill the critical functions that system abort provides today, but will do so in a manner that minimizes the impact to the business applications present on the system.

Software Resiliency Tactics

The model is useful in understanding how the individual efforts that are being employed fit together within the software resiliency framework. Below is a brief discussion of some of them and explanation of how they fit into the medical model.

Self-Managing Systems -- Resource Control

Some system aborts are a result of the depletion of critical system resources. Expanding table limits eases this problem, but the highest level of resiliency is obtained through monitoring critical resources and taking action to prevent applications from depleting them entirely. This is a form of preventive medicine.

A resource monitor is being developed which will take action when a critical resource reaches a dangerously low level. Operator notification will be given and the system will go into a degraded mode where logons are prohibited and process creation is limited. When the crisis is over and resources are again adequate, the system will automatically resume full operation.

Logging and Software Predictive Support

A new logging facility for internal use is being developed. Logging of information will be crucial to problem diagnosis for exception conditions that do not result in a system abort.

There will also be opportunities for predictive types of functions. Log records of specific software events could be analyzed and used to notify operations of impending problems while there is time to correct them.

Try/Recover Guidelines

The HP Pascal language provides a try/recover construct that allows a procedure to control the logic flow even when an unexpected trap occurs. When a trap occurs in the try block of a procedure, control is transferred to the recover block where appropriate action can be taken. Guidelines on the use of this construct have been developed.

Among other things these guidelines show how a procedure can:

1) Keep track of resources collected so they can be released if an error occurs. The releasing of resources is often vital to the recovery or cure of a problem situation.

2) Make sure errors and traps are caught at the appropriate level where they can be handled properly. Early detection of errors or potential contamination conditions provides a better chance of handling them without the use of a system abort.

Locking

Synchronization guidelines have been developed that will help engineers with such problems as making sure the right lock is held to perform a given operation and making sure that proper locking sequences are followed. This form of preventive medicine will avoid corruption and reduce the likelihood of a deadlock.

Tools

Tools are are being developed that will perform syntactic analysis on system code. This will provide an idea of the current resiliency levels or "health" of different parts of the system. A data base will track the "medical history" and improvement of modules. When a procedure is visited for defect repair or enhancements, it will be strengthened using guidelines that have been developed.

Tools are being developed that can be used like "surgical instruments" that will allow us to go in and repair or diagnose damage that has previously been contained, while the system is up.

Applying the Model to Modules and Operations

Given the fundamental functions that system abort provides, the following questions are being asked about modules in the operating system.

1. What is the detection strategy?
2. What is the containment strategy?
3. What is the diagnosis strategy?
4. What is the recovery strategy?

In cases where the answer is system abort, target strategies are being developed to allow for the functions to be served without bringing the system down.

Classifying System Aborts

As previously mentioned, the medical model is useful in the classification of system aborts. Are there system abort calls present that do not fit within the model presented? If so, can the model be expanded to include and classify them? Can some classes be easily eliminated?

For example, there are places where system abort is called because of an unexpected escape. These system aborts could be classified as the result of a poor detection strategy. By the time the error is detected, the cause is lost and the only safe way to ensure containment is to system abort. The try/recover guidelines are being applied in these cases to strengthen the error detection and allow the containment to be accomplished with less impact to applications.

Application Isolation - Quarantine Strategies

Application isolation involves the concept that when a particular application encounters a problem, only it is affected. Other applications on the system remain operational. We are developing quarantine strategies that will allow problems to be isolated to specific files, volumes, or volume sets, instead of affecting the entire system.

This containment technique will allow for the abort of application specific operations in order to prevent contamination of user data, but will do so in a manner that affects only that application using a particular file or file set.

For example, while it is not possible to disable a critical function of the operating system like file label management, is is possible to disable access to a given file label, a given label table or to disable label allocation on a given volume. For example, if corruption were detected in a specific volume's label table, it would be quarantined, thus preventing the contamination from spreading to the application or its data, but the system would stay up.

Along with this approach, proper diagnostic and recovery mechanisms are also being developed.

Rollout Plan

Research into the STARS data base, analysis of patches, and several discussions with support engineers have identified the kernel and the file system as the main areas in which we should concentrate our efforts. Further investigation will determine some specific vertical paths on which we will concentrate our system abort elimination efforts. (A vertical path is an operation involving several operating system modules, the file open path for example.)

In addition we are looking at a phased approach to replacing the functions that system abort provides. Containment, in general, is much easier than recovery. Therefore, as a first phase of the strategy, we can concentrate on quarantine strategies that involve disabling access to certain portions of the system. Containment strategies, balanced with adequate diagnosis functions, can be developed first. Recovery from the condition might still be somewhat severe. For example, a volume set is disabled without crashing the system, but a reboot is necessary to recover it. The next phase would be to develop tools and automatic recovery strategies to limit the impact of recovery.

One example of containment is the KSAM enhancement that was made for release 4.0. Before this release internal corruption in the control block of the KSAM file as well as unexpected escapes caught by the type manager would cause a system abort. Try/recover blocks were added around sensitive calculations dealing with control block verification. Internal routines were strengthened with try/recover blocks so known errors were passed to the type manager thus avoiding most unexpected escapes. Operations on the file that would have previously led to a system abort now just return errors, increasing system uptime. The file itself remains available for on-line diagnosis or may be stored to tape for later analysis.

Processes for Continuous Improvement

The HP 3000 lab has instituted processes designed to provide continuous improvement in the quality of operating system software. A formal process called Root Cause Analysis (RCA) has been implemented. RCA is a process by which software defects are analyzed by asking why they occurred. The "why" question is asked at least five times for any given defect in order to determine the true root cause and in what development phase the defect was introduced.

The results of the analysis pinpoints phases of development where the most defects are being introduced and where a more rigorous and formal inspection processes can be applied. RCA is an ongoing activity and includes a set of metrics by which progress can be measured. It also has built in feedback mechanisms that allow for improvements in the process itself. This represents another form of preventive medicine.

As part of the process for continuous improvement, the HP 3000 internal architect team has updated the specifications for design and code documents that are part of the HP 3000 software lifecycle. New sections that specifically address software resiliency issues have been added. Adherence to resiliency guidelines and error handling standards are part of the checklist items that need to be addressed as part of the exit criteria for certain phases of the software lifecycle.

The software resiliency team is engaged in ongoing consultations with various module owners and developers to assess current abort strategies for modules and procedures in some specific vertical paths and to develop target abort strategies.

HP: The Best Alternative to Mainframe Computing (... and Getting There)

by Robert Winter

The New Demands

Introduction

For many years Information Technology (IT) executives and managers have acknowledged the need for more cost-effective computing than mainframes. Unfortunately, acceptable alternatives to mainframe-based computing have been lacking--until now. Today, many IT executives view that future IT environments must be user-based. With the rapid emergence of technologies like client/server, the question is how to get to there?

A brief history...

A great number of current mainframe environments are based upon hardware and software philosophies that arose in the late 1950's and 1960's. Data processing was the domain of small groups of 'elite' technicians, while users had little or no access to the computer. The 1960's and 1970's gave us progressively larger batch processing computer systems that emphasized the automation of repetitive tasks to process progressively larger amounts of data. Computing was still tied inextricably to mainframes, but slowly the pool of users with access was broadening as the cost of computing came down.

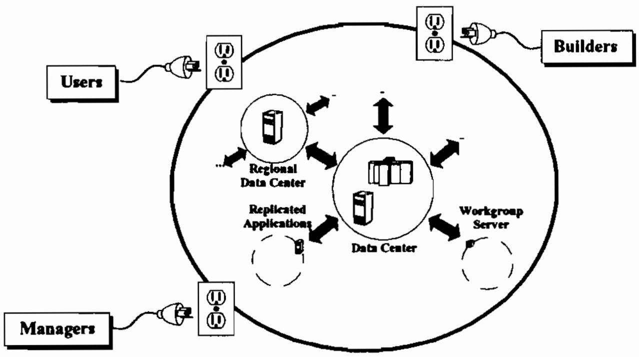
The Emergence of a "New" Data Center

Mainframe-based	User-based
MIS access	User access
Closed	Open
70's centralized approach	90's client/server approach
Data center	Desktop to data center
>10 support staff	1-2 support staff
Big room, water cooled	File cabinet, air cooled
> $3 million, CISC	< $1 million, RISC

The late 1970's and the 1980's saw the emergence of "mini" or midrange computers--multi-user systems that brought computer resources to the department level and closer to the user. Additionally, midrange computers did not require special environmentals, e.g., water cooling. The early 1980's saw the emergence of the personal computer (PC). PCs furthered the evolution towards decentralized computing. PCs could act as terminals to a host and shuttle data back and forth. The empowerment of the user began.

In the 1990s with the emergence of powerful RISC-based systems that scale from the desktop to the data center and powerful, new software or "middleware" (where systems share the work of data processing each doing what they do best), we see the realization of user-based computing accelerating. Users, whether managers, builders of applications, or end-users, will have the ability to cost-effectively and transparently access information across the entire enterprise.

HP: Cooperative Computing for the User



As user-based computing grows, mainframe-based data centers will undergo dramatic changes. Gartner Group predicts *"...the mainframe segment of the On-line Transaction Processing (OLTP) market will decrease from 55% to 43% by 1993."* Additionally, over the last three years the number of MVS sites has declined by 18%. Mostly due to support of "legacy" or older applications, mainframes will be here for quite some time; however, their market share decrease is only beginning.

<u>The economic shift</u>

Five powerful and accelerating business forces will recast the role of information technology (IT).

- **Worldwide competition** demands absolute improvements in an organization's cost structure, quality and service. Time to market must be compressed dramatically, and that requires much greater shared information across corporate boundaries. Heightened global competition also means tighter budgets. IS organizations must be more aligned with the business.

- **Organizational flattening** makes companies leaner, more cost-effective and more responsive to customer and market needs. As management styles change to support flatter organizations, IS will have to provide a denser, richer communication and IT infrastructure to all users. Being user focused is key to any IT success in the future. According to IDC, 1992 *"...the relative influence of MIS in purchase decisions has dwindled from 85% in the 1960s to 40% in the early 1990s."*

- **Economic, political and technological unpredictability** necessitates flexible business units that can respond rapidly to change. IS must be able to facilitate change. Standards, particularly interoperability of networks, databases and applications, and modularity of systems will enable flexible business restructuring upon demand.

- **A focus on core business competencies** will be critical to the sustained success of an organization. It means companies must concentrate on what they do best. Increasingly, organizations are forming alliances to complement their core competencies--partnerships, strategic suppliers, etc. generating the need for new levels of inter-enterprise interfaces to support external data exchange and business transactions. For example, a customer transaction at a car rental company may also coordinate with the promotions or products of other companies based on the customer's past travel-related buying patterns.

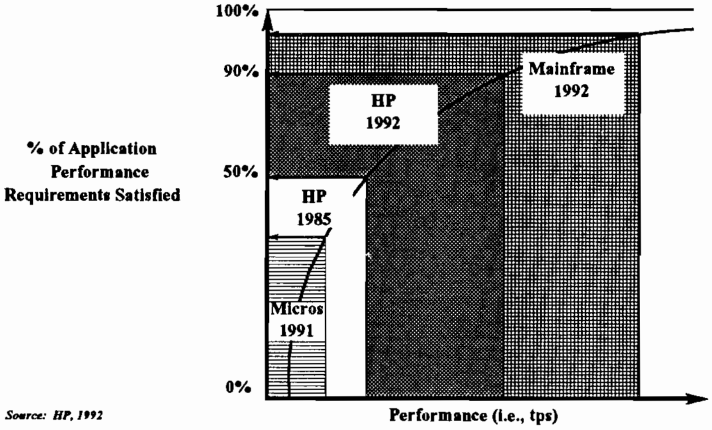
- **Customer service and satisfaction is** becoming the new battleground for competitive advantage. As the example above shows, greater focus on the customer can mean more revenue, more revenue per transaction and greater customer satisfaction that enhances customer loyalty. Since the end user is the interface with the customer, it is important for IS to have an aggressive stance to user-based computing.

Many progressive enterprises are adopting cooperative computing information architectures to effectively manage these shifts. While many still use mainframe-based computing, newer, more strategically-based systems are user-based--adopting in many cases HP computer systems. With huge investments in current computing assets, any step forward has to protect those investments. The ability to co-exist in this fashion while supporting the evolution to more cost-effective, user-based computing is HP's vision, direction and delivery. This process is allowing customers to re-invent their business in an evolutionary fashion.

The technology shift

Access to new, more cost-effective technology is inspiring the shift from mainframe-based to user-based computing. In a June 4, 1990, Computerworld survey (194 IS Chiefs), *"48% of the respondents said they were considering mainframe alternatives."* Greater flexibility, reduced cost and greater user access were recognized as the principal benefits. As little as two year later, a 1992 Forrester survey shows this trend is rapidly accelerating, *"...a staggering 80% of the 75 Fortune 1000 firms interviewed are looking to break the mainframe's grip on application processing."*

The Gap is closing between the Midrange and Mainframes



Technology issues accelerating interest in mainframe alternatives include:

- **Older mainframe models are prohibitively expensive.** There are well over 20,000 43XX, 308X and low-end 3090 mainframes that are not cost-effective and are lagging technological advancements. Conversely, midrange systems, like the HP 3000/9000, have cost-effectively grown in performance and capabilities. HP's newest class of systems provide similar performance to IBM's 3090-600J (their largest system shipping as of early-1991...only a 18 month lag) at 1/5 the cost...the gap is closing!

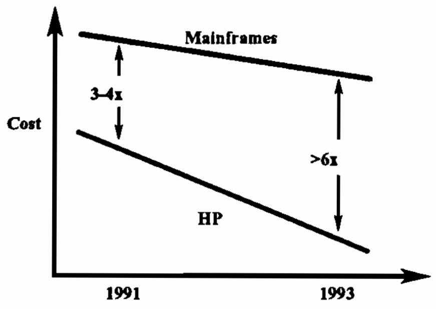
  Gartner Group/LCS, 1991 predicts that intermediate mainframe shops will increasingly seek alternatives. *"...The effect of compound growth rates over the last five years has created a wider disparity between 'large' IBM mainframe shops and 'intermediate' mainframe shops. As most of the new functionality focuses on the large shops, the widening performance band increasingly exposes IBM to losing its intermediate-size shops. Once lost to back-level machines (43XX, 308X, 3090-1XX/ 2XX/3XX), it will be exceedingly difficult for IBM to shoehorn new technology into* *user budgets. Without differentiation and out of the IBM mainstream, intermediate-size shops will increasingly turn to alternatives."*

- **Many mainframe-based applications sorely need updating.** These applications either represent aged technology--primarily batch processing with limited OLTP or simply can not meet today's changing business requirements. In contrast, early adopters of client/server computing, by only providing a graphical user interface (GUI) client, *"...have seen*

  >35% more tasks completed,
  >16% fewer errors,
  >51% lower frustration and
  >23% more tasks attempted"

  (Source: Temple Barker & Sloan, GUI Research Study, 1991). More importantly, by making the proper capital investment today, early adopters are best positioned to competitively meet tomorrow's business needs. As well, they can evolve towards a more complete client/server topology for even greater productivity gains with no or little new capital investment.

In the Future, the Cost Gap will Rapidly Grow



Source: HP, 1992

...Mainframes falling further behind

Additionally, in 1992 many client/server based applications are coming to the market on HP platforms--SAP, Dun & Bradstreet, PeopleSoft, Datalogix, FOURTH SHIFT, Collier-Jackson, Mitchell Humphrey, DRC, Software AG, etc. Further, application vendors are focusing on traditional midrange platforms. Gartner Group, 1991 suggest *"...there is a 70% probability* (the highest of any platform) *that midrange* *systems will evolve to become Enterprise Server Platforms (ESPs)."* Many of the reasons cited include investments HP is making in ...

- open systems,
- next generation "middleware,"
- scalability from the desktop to the data center,
- lower cost of processing,
- flexibility in application deployment and
- development of enterprise-class resource management.

- **New technologies are robust enough to handle many enterprise-class applications.** Functionality, availability, serviceability and tools (FAST) have grown dramatically for HP systems--whether it's managing the resources of the systems (OpenView, backup/restore, scheduling, printing, etc.), providing high availability (mirrored disk, disaster tolerant configurations, etc.) or enhanced enterprise-class service and support. Additionally, the standards "middleware" (DCE, DME, DOMF, OpenODB, Transarc/Encina, etc.) that is being driven by vendors like HP will truly enable user-based computing. This "middleware" will allow IT to become a truly integral part of the organization. It will provide for flexible, transparent, available access to information across the enterprise in a secure manageable manner while ensuring data integrity and performance scalability from the desktop to the data center.

The Process: How To Get There From Here

What is "rightsizing," "downsizing" or "mainframe alternative?"

HP has "coined" the term mainframe alternative. It is used as an umbrella term to describe:

a. **"offloading"** (typically from a 3090-3XX+ class system) an application(s) to a more economical HP system,
b. **"replacing"** of a mainframe outright (typically a 43XX, 308X or 3090-1XX/2XX/3XX class system),
c. **"downsizing"** to a smaller, replicated systems, or
d. **"rightsizing"** to a system of the appropriate capacity.

In all cases the motivation is driven by the economic and technological shifts discussed earlier. Further, mainframe alternatives may include smaller and/or larger systems, and centralized and/or distributed systems; the common tread is that the solution has superior economics, both tangible and intangible.

If we use "offloading" and "replacement" as representing the definition of mainframe alternatives, HP, as of late 1991, had well over 100 successful wins. Roughly 40% were "offloads," while 60% were "replacements." Additionally, significant numbers were new customers to HP.

Reshaping the business

Business re-engineering has been one of the most broadly talked about concepts of the past few years throughout the business community. A survey by Index Group, Inc. 1991 indicates, *"...reshaping business processes has vaulted to become the No. 1 concern of 243 U.S. and Canadian IS executives."* Enterprises are starting to refocus on fundamentally changing the way they approach traditional business functions. Being user-focused is key. The study goes on to suggest that *"...IS executives strongly believe that more user involvement in systems specification is needed. An overwhelming majority, 87% in Europe and 83% in North America, agreed that getting the systems requirements right is the key to producing successful IS."*

HP has seen customers use "downsizing" or "rightsizing" as a way to pursue reshaping their business processes to gain many of the business benefits discussed earlier. The following is a list of re-engineering principles summarized from an article by Michael Hammer in *Harvard Business Review* that many HP customers have successfully employed.

1. **Organize around the outcome, not the tasks.** As we specify application functionality and technology alternatives, a first step should be the evaluation of the underlying business processes. In this evaluation, organizations, departments and

workgroups should be reorganized to simplify processes and streamline outputs rather than tasks.

2. **Have those who use the output of the process perform the process.** A key to improvement is to empower those in a position to enhance the process. Wherever possible, management should empower the individual.

3. **Subsume information-processing work into the real work that produces the information.** The information automation model must reduce the passing of tasks between departments by a common information repository/database. Today paper is passed between departments for sign-off and processing. These processes reduce operating efficiency and can be eliminated with integrated systems.

4. **Treat geographically dispersed resources as though they were centralized.** Decentralization as a business theme for manufacturing should be modified to 'decentralization to the appropriate level.' Guidelines and standards should be established by cross-divisional teams with the primary motivation being to reduce the burden rather than adding to it--this is the rule of 'economies of scale.'

5. **Link parallel activities instead of integrating their results.** In manufacturing, this principle is being broadly applied in engineering as concurrent engineering.

6. **Put the decision point where the work is performed and build control into the process.** For several years now, manufacturing companies have used this principle to incorporate a quality ethic into the process rather than inspecting the final product. This principle again reiterates the need to empower the individual with information and decision-making capability.

7. **Capture information once at the source.** The key to the success of this endeavor is the establishment of a common core information technology (IT) architecture among the functional areas (e.g. sales, finance, etc.).

In each of these principles, the change proposed can dramatically improve overall operating performance. However, these suggested changes are often radical, difficult to adopt and virtually impossible to justify with traditional financial analysis. Therefore, many mangers use reducing IT operating cost and long-term capital expenditures (mainframe alternative) to provide the tangible financial justification.

<u>The steps...moving from the mainframe to HP</u>

- **Step 1. Assess the status quo.** The following are a few considerations to examine:

  General
  - Is the organization under profitability and competitive pressures?
  - Is senior management educated as to how IT is or can be a strategic part of the business?

- What is the degree of management and user satisfaction?
- Would the business benefit from better functionality and improved response time?

Cost
- IS organization budget pressures?
- Is it time to upgrade or renew a system lease?
- How significant are your support/maintenance costs for hardware and software against those of a comparable midrange computer (e.g. HP 3000/9000)?
- How do your software license fees compare against those of a comparable midrange computer?
- Are over 30% of your programming staff dedicated to maintenance of current systems?
- How do the cost-per-MIP or $-per-transaction compare?
- What is the cost of maintaining your current computer facility, including power usage, cooling units, cost per square foot? It an expansion of the physical facility needed?

Application
- How old or weak are applications?
- How large is your development backlog and how productive is your current development staff?
- Do future applications call for client/server or distributed computing?
- What are the criteria in determining which applications to offload or replace first?
- Does compiling programs compete with production resources?
- Are your systems mainly batch?

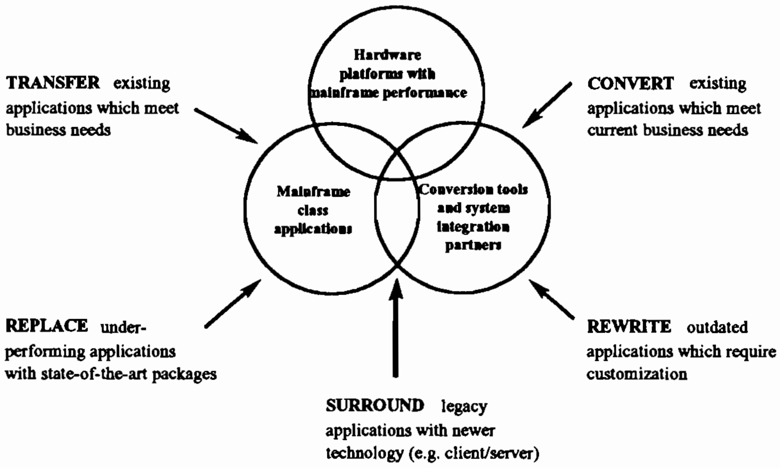
- **Step 2. Develop an information architecture from the bottom-up.** Where is data generated? What functions must be performed on it? Where must the information go?

- **Step 3. Evaluate your information architecture from an objective viewpoint.** How can it give your enterprise an operational advantage over the competition (reshaped business processes)? How can it lower cost/increase functionality? What are the priorities for change?

- **Step 4. Gain top-level management commitment.** This is important. What are the tangible benefits (e.g. IT ROI)? What are the intangible benefits (e.g. business processes' improvements)? Are the "users" involved (remember there are alot of politics in these transitions and the users can be strong allies)?

- **Step 5. Assess your migration strategy.** When moving applications from a mainframe there are five fundamental approaches. Any strategy many include one or all of the approaches.

HP's Mainframe Alternative Solution: Five Implementation Methods


Hardware platforms with mainframe performance
Mainframe class applications
Conversion tools and system integration partners
TRANSFER existing applications which meet business needs
CONVERT existing applications which meet current business needs
REPLACE under-performing applications with state-of-the-art packages
REWRITE outdated applications which require customization
SURROUND legacy applications with newer technology (e.g. client/server)

- **Transfer.** Many mainframe software providers have ported their products to the HP environment. Application vendors include, but are not limited to, IBI, Software AG, CINCOM, Dun & Bradstreet, SAP, SAS, Computer Associates (CA), PeopleSoft, Lawson and Cyborg. These vendors' applications and associated data can be relatively easily ported to HP systems to reap cost and quality benefits.

- **Convert.** Flat files, i.e. VSAM, can it migrate over as is. Relational databases, as well, provide a relatively straight forward migration, particularly when DB2 or SQL/DS databases on the mainframe have kept their adherence to the ANSI SQL standard. CICS/COBOL, which represents roughly 83% on all OLTP code, needs to be converted. HP partners, Infosoft, Integris, Zortec (Unisys), VI Systems and Jacksonville Software provide conversion tools. Partners like Infosoft have a relationship with IISI, a system integrator (see Integrators below) to provide a "one-stop" migration/conversion services. Additionally, other tools exist to ease the migration from mainframes to HP.

- **Replace.** In many cases the mainframe applications are outdated and require a high level of maintenance or have large backlogs of enhancements requested. HP has many enterprise-class solutions that have in many cases superior functionality (e.g. ASK, Datalogix, DRC, BSA, Mitchell Humphrey, Collier-Jackson, VTLS, Inlex, Uniplex, R&D, Brock, Fourgen).

- **Rewrite.** Customers with customized mainframe applications frequently have chosen to rewrite their applications to take advantage of new functionality, business re-engineering or new technology (e.g. client/server).

- **Surround.** Some organizations face a dilemma, they realize their mainframes are not cost-effective but their outdated applications are deeply entrenched making migration difficult. In these cases, many choose to surround the systems and applications with newer technology to add new functionality. This is usually based on the client/server model involving HP 3000s or HP 9000 Series 800s as servers, and/or HP 9000 Series 700 workstations as high-function clients.

- **Step 6. Assess the technology change.** Whether centralized or distributed, have you determined the rightsize system(s)? Disk space requirements (HP systems tend to use 30% less disk than mainframes for the same amount of data)? Can you move away from heavy dependence on tape processing (HP systems support devices like 3480/3490s; however, it is typically used for backup or transferring data to/from another computer)? Where do the 3270s go (PCs in many cases have better economies than 3270s due to their high maintenance cost)? How many terminal/client devices are needed (HP systems are highly interactive and the number of users tends to go up)? How are existing print jobs going to be supported (HP provides printing based on industry standard PCL from the desktop to the data center; additionally, firms like IDATA provide conversion tools from IBM's AFP and OGL to PCL)?

- **Step 7. Select partners.**

  - System provider/project manager (HP).
  - Software solutions (applications, 4GLs, RDBMS, etc.).
  - Application/system consultants (Innovative Information Systems (IISI), Cap Gemini, Andersen Consulting, etc.).
  - Migration/conversion tools (Infosoft, Zortec, VI Systems, Jacksonville Software, etc.).

- **Step 8. Develop project plan.** What resources (e.g. staff, data structure and flow, inventory of screens, reports, files, processes) are available? Timeframe? ROI? Pilot or test bed project? How will change and re-training be dealt with? What's the contingency plan? Does your plan include the "user"?

- **Step 9. Execute.** That's right ... just do it. This is too large an opportunity not to gain the business results and the success that come with it. Work with technical consultants from HP's professional services organization and our partners to smooth the transition.

<u>The challenges</u>

The process of moving from the mainframe is not a simple one-to-one movement. Remember this is hard work, it is as much a technical and managerial art as a science.

- The Politics and the People. Computer professionals tend to become emotionally attached to technology they know, often making a rational alternative evaluation hard. The very factors that make an alternative, like HP, an attractive business decision also make it difficult at times to sell and implement...

  A. Advanced architectural design. The advanced architecture (e.g., PA-RISC, single-level storage) of HP computers makes them different from mainframes. An effort is needed to understand the characteristics of HP systems and how they can benefit an organization.

  B. Programmer productivity. Programmers support the concept of higher productivity--in the abstract. But when it threatens to disrupt their lives, they often find subtle ways to resist. HP systems can provide much greater productivity, via products like HP's Softbench. This can mean more and better applications for the enterprise.

  C. Ease of use. Systems like those from HP do not require the technical software specialist that are necessary in the mainframe environment. The mainframe environment is complex and presents IS management with a steady stream of difficult decisions. Mainframe professionals tend to feel that they are a necessary part of a complex process. However, the ease of use of HP systems can afford the IS professional the opportunity to gain new, broader skills, e.g., in understanding the business' and the user's needs.

  D. Reduced costs and staffing. Many managers take pride in the size of their staff and budget. Often companies, historically, set pay and perk levels based on these criteria. A reduction in IS budget and staff is therefore not always welcome news to IS managers. The reality, based on successful implementations of HP solutions, is that this affords the organization to more effectively investment in new areas. Those areas might be engineering, customer service, etc. However, the IS organization gains since the mainframe alternative solution proves to be much more user-based and cost-effective.

  How can senior management insure a mainframe alternative (e.g., HP) is given appropriate consideration?

  1. Involve IS management in corporate planning. If IS management is allowed to participate in the planning process for the organization, it is more likely to think in terms of what is best for the organization as a whole.

  2. Align IS goals with organizational priorities. Management must insure that IS is focused on solving problems for other functional departments. IS should be measured on the cost and quality of the solutions it provides and on the

satisfaction level of users it serves. The goals of IS should be stated in business and not technical terms.

3. Form a steering committee. Many organizations have an executive steering committee that includes the managers of key user departments. This committee acts to a board of directors for the IS function. Such steering committees insure that IS is focused on solving organization problems rather than on dealing strictly with technical issues.

4. Reward true cost savings. IS management must be provided with positive incentives to improve their productivity. Many organizations actually indirectly punish managers who reduce costs by basing compensation on budget and headcount. Senior management must be prepared to reward IS management directly for improving its productivity.

5. Reducing the bias in the evaluation. No one is going to favor an approach that saves money if their own salary forms part of the savings. The mainframe alternative must be evaluated in a way that will prevent individuals who feel threatened from controlling the evaluation.

6. Develop approaches to overcome technical staff resistance. Some personnel policies and strategies that have been successfully used:

   - No layoff assurances. It is usually possible to accomplish any staff reduction goals through attrition. It can help to assure key technical personnel at the start of the evaluation that they will not be laid off. HP has seen that most customers re-deploy some staff to user groups or applications development.

   - Fixed term agreements with bonuses. A migration from mainframes to alternative systems can take months and sometimes longer. Many organizations starting a migration from a mainframe to HP have identified key people needed to support the mainframe and have offered them special bonus incentives to stay until the migration is complete.

   - Guaranteed retraining. Computer professionals are usually concerned about building their professional skills. A commitment to train key personnel as part of the plan for migration can reduce the risk of losing them. With HP's focus on delivering open systems this can actually enhance their marketable skills.

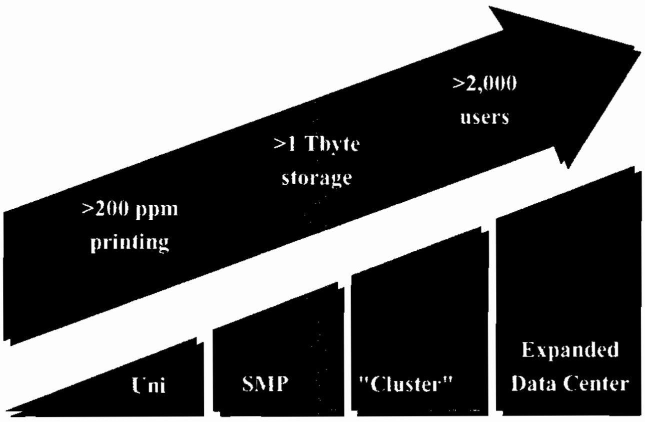
- Determining the Role of the Client and the Role of the Server. The following explores some of the pros and cons of each role.

C/S Style	**Efficiency**	**Integrity**	**"Evolvability"**	**Manageability**
Host/Server	Higher	Higher	Lower	Higher
Desktop	Lower	Neutral	Higher	Lower

The desktop or client (e.g., Windows, Motif based) role provides for greater "evolvability." For example, a workstation (e.g., PCs, HP 9000 Series 700s) in terminal emulation-mode provides productivity gains (see The technology shift section) while also providing the infrastructure from which more complete client/server applications can be implemented "down the road" without necessarily making additional capital investments. Additionally, workstations have proven very effective in offloading decision support, office automation and computer-aided design/engineering/ manufacturing (CAE/D/M) applications from the mainframe to the desktop. The mainframe provides very expensive performance, and a greater deal of functionality and complexity not needed in these areas.

Conversely, centralized control is critical if the efficiency, integrity, and manageability of the data are to be optimized. This is the role of the server (e.g., MPE/iX, HP-UX based). Further, it's easier to maintain (more cost-effective, etc.) one or a few host-based systems than dozens or hundreds of desktop-based systems. For these reasons HP 3000s and 9000 Series 800s are more ideally targeted for "mission-critical" applications. Applications which serve core business functions. The value is that each (client and server) provide complementary strengths and much more cost-effective user-based computing.

Scaling for Large Configurations



HP: the most effective FAST mainframe alternative program

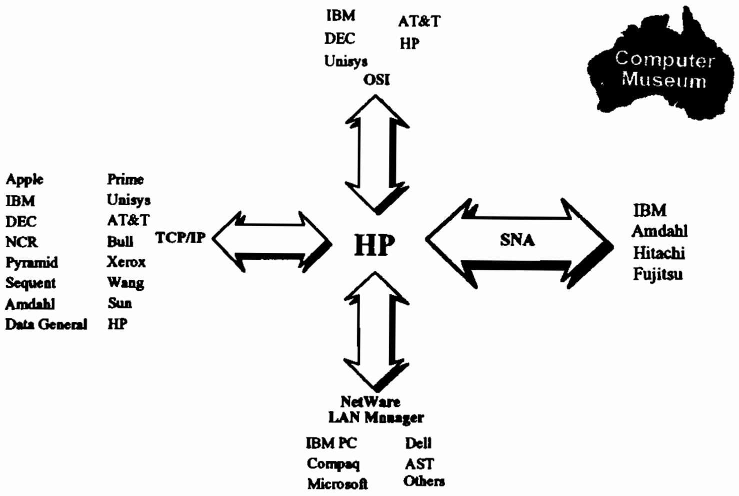
Why is HP the best mainframe alternative? Because, HP has the best and most cost-effective FAST program to support user-based computing. FAST is Functionality (e.g.

performance, scalability, systems management, configurability, networking), Availability, Service (service, support, maintenance and integrators) and Tools (e.g. conversion/ migration, CASE/4GLs, databases, application solutions, information access).

Functionality Availability Service Tools

- Performance. Two recent benchmarks on a representative HP system, the HP 3000 Series 980/100, have shown superior performance relative to water-cooled mainframes.
  1. A batch benchmark showed a Series 980/100 (14.5 minutes elapsed time) to be similar in performance to an IBM 3090-180J (15.6 minutes elapsed time) running MVS/ESA (the benchmark involved the batch processing of 1 million/400 byte order processing records).
  2. A client/server OLTP benchmark (PeopleSoft/HRMS application) showed a Series 980/100 (8,000 transactions per hour) provided 50% MORE throughput than an Amdahl 5990, equivalent to an IBM 3090-200J (5,360 transaction per hour), a two-way multiprocessor running MVS/ESA.
  3. At growth rate in excess of 75% per year, HP's newest high-end systems (shipping mid-1992) will provide at least similar performance to an IBM 3090-600J (their largest water-cooled system shipping as of mid-1991) at less than a 1/5th the cost!
- Scalability. HP offers the broadest scalable architecture. This enables applications to run on the desktop up through the data center. Further, scalability facilitates the control and manageability of implementing client/server applications across the enterprise.

A Natural Fit for Any Data Center Environment

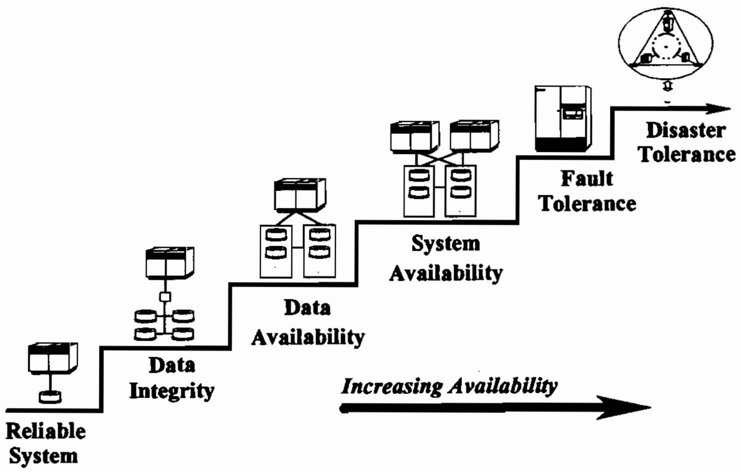


- Systems Management. HP provides a broad array of products and services to management computer resources whether local or remote, whether desktop or data center. HP OpenView (the framework for OSF's Distributed Management Environment) provides an integrated environment from which systems, networks and application resources can be managed. The objective is to provide a "lights-out" type environment: minimizing staff costs, increasing service quality/ availability and increasing the IT architectural flexibility. Further, mainframe-class solutions from vendors like Computer Associates (CA) on the HP 9000 Series 800s offer a high degree of functionality in the UNIX environment while minimizing potential retraining.
- Configurability. HP provides for the ability to scale printing from the desktop (LaserJet) to the data center (HP 5000) or from 4ppm to over 200ppm in a production environment while supporting the *de facto* printer control language (PCL). Storage capacities by the end of 1992 will exceed 1 Tbyte for HP systems, while supporting well over 4,000 terminal sessions.
- Networking. With complete support for SAA/SNA, HP provides strong interoperability and coexistence capabilities with mainframes. Additionally, broad support for various network standards exists--TCP/IP, OSI, Netware/LAN Manager, etc.

Functionality **Availability** Service Tools

- Reliability. HP systems provide mean-time-between-failures (MTBFs) of over 4 years. Standard disk storage subsystems exceed MTBFs of 20 years. With the operating systems, higher degrees of "bullet proofing" will occur. For example, the mid-1992 introduction of MPE/iX provides for application isolation and minimized system failures due to application aborts. MPE/iX in a 1991 Prognostics survey showed reliability of *"...roughly one outage per year per system on average."* This level of reliability is similar to IBM's MVS environment. Repeatedly, surveys by organizations like Datapro, Computerworld have rated HP as the No. 1 quality computer manufacturer.
- High Availability. A broad portfolio of products exists to increase the availability of systems to even higher degrees--providing availability approaching 99.9%. Products include disk mirroring, processor switchover, etc.
- Fault Tolerance. The HP 9000 Series 1200 offers true fault tolerant capabilities by providing complete redundancy in its components. Additionally, as HP's newest high-end systems (mid-1992) evolve over the next few years, fault resilient capabilities will be brought "on-line." These features will include fault tolerant memory arrays, redundant power supplies, on-line component repair/replacement, etc.
- Disaster Tolerance. The HP 3000 offers the facility, Quest/Netbase, to provide both "sysplex-like" and "expanded data center" capabilities. Systems can be placed in geographically remote locations with the assurance that in the event of a disaster at one location the user (transparently) will not be impacted. Additional benefits include load balancing and horizontal performance and configurability growth.

High Availability in the Data Center



Functionality Availability **Service** Tools

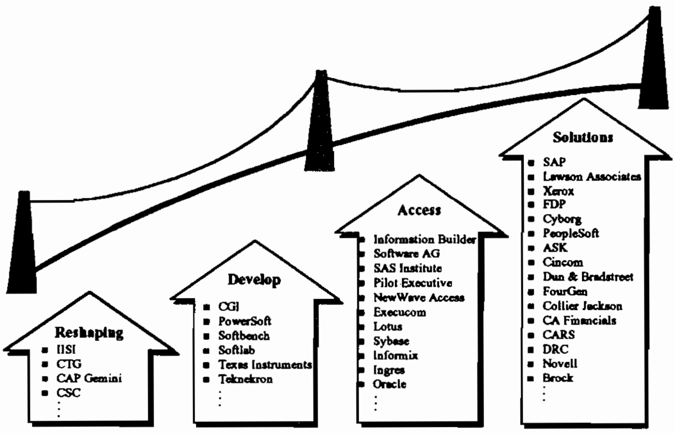
- Premium support. By late-1992, a new level of 'proactive' support and maintenance will have been ushered into HP environments. This provides complete coverage through a single comprehensive contract and annual account planning. The support "partnership" will provide on-site specialists, personalized service and "business focused" support.
- Enhanced breadth of solutions. Additionally, expanded support to 24 x 7 coverage is available. Additional solutions include disaster recovery planning, backup services, performance tools/consulting, etc. From a proactive perspective, predictive hardware and software support will allow better problem resolution before it happens. As an example, "hooks" within MPE/iX provide for the recording and reporting of its status. This allows potential problems to be more proactively addressed.
- Integrators. HP has had a great deal of success partnering with system integrators to support customers migrations/conversions from mainframes to HP. Most are 'boutique' firms that have successfully focused in this area; integrators include Innovative Information Systems, Inc. (IISI) in Norwood, MA; Andersen Consulting; CAP Gemini; Wesson, Taylor and Wells; CT Partners and SAIC.

Functionality Availability Service **Tools**

- Conversion/migration tools. Many of these were discussed in earlier sections.
- CASE/4GLs. Along with enterprise-class CASE and 4GL tools, many of which are first generation client/server, on the HP 3000/9000 (e.g., TI, Softlabs).

- Databases. Database support includes HP's Allbase, Oracle, Ingres, Sybase, Informix, Progress and Unify.
- Application solutions. HP is providing many traditional mainframe as well as newer client/server applications--allowing customers to bridge.
- Information access. A broad offering of tools that can access production data are available--SAS, IBI, Pilot Executive, Lotus, Execucom, NewWave Office, Express, HP Desk, HP Openmail, etc. Informative access on HP systems has been and is viewed to be much easier and more responsive than traditional mainframe environments.

HP Systems--Bridging from the Mainframe to the User



The Results

The ROI

"The average IT operations savings reaped by account is $1.7 million per year" (Source: Forrester, 1992; HP, 1991).

The following representative company illustrates these potential savings. ABC, Inc. (due to the case specifics discussed the name has been changed) found themselves facing a significant computer upgrade situation in early 1991. Their existing IBM 4381 would reach full capacity in six months and their HP systems in 18 months. Their decision had to be cost-effective, cater to their computing requirements over the next 5 years and provide them with an easy growth path. After an investigation, ABC, Inc. narrowed their choice to one of two options:

1. Upgrade the IBM 4381 now and the HP systems in two years time. Upgrade the IBM 4381 again in five years time to an IBM 3090-200J (or equivalent), or

2. Upgrade the HP system immediately and migrate applications running on the IBM 4381 onto the HP system. In five years time, upgrade the HP system.

Option 1: Maintain and upgrade existing IBM and HP systems

	Year 1	Year 2	Year 3	Year 4	Year 5
Lease for IBM hardware including upgrades	228,702	372,160	410,085	448,010	760,505
IBM hardware and software maintenance costs	581,967	611,068	641,622	673,703	781,125
Lease for HP hardware including upgrade and additional HP software	73,980	90,973	97,958	122,722	129,707
HP hardware and software costs	128,070	134,475	141,197	148,275	155,670
Facilities and operations cost	1,492,855	1,581,748	1,675,945	1,775,757	1,881,518
Total Yearly Costs	**2,505,574**	**2,790,424**	**2,966,807**	**3,208,467**	**3,708,525**

Option 2: Migrate IBM applications to an upgraded HP system

	Year 1	Year 2	Year 3	Year 4	Year 5
9 month lease for IBM 4381 during migration phase	171,527				
9 month IBM hardware and software maintenance costs	436,477				
Lease for HP hardware incl. upgrade	344,272	334,170	342,897	368,998	451,403
Lease for existing HP hardware (peripherals, etc.)	50,363	50,363	50,363	50,363	50,363
HP hardware and software maintenance costs	191,432	201,003	211,052	221,607	232,685
Facilities and operations cost	1,193,945	1,262,410	1,334,977	1,411,888	1,493,407
Conversion of IBM applications (incl. training, etc.)	456,752				
Sale of existing IBM 4381	(181,818)				
Total Yearly Costs	**2,662,950**	**1,847,946**	**1,939,289**	**2,052,856**	**2,227,857**

Net Yearly Savings Option 2 versus Option 1	**(157,376)**	**942,478**	**1,027,518**	**1,155,611**	**1,480,668**

The second of the two options clearly showed that migrating the IBM applications to an HP system was the most cost-effective option. ABC, Inc. total net savings over the five year period will amount to almost $4.5 million with an expected payback period of 31 months. The following further exemplifies the tangible IT benefits HP mainframe alternatives can provide:

- Excellon goal was to *"...reduce the IT budget from 2.5% of sales to 1.5% with no compromise in product quality or customer service."* With HP 3000s they did.

- General Chemical reduced their *"...IT operations costs by over 70% ... this translates into $2 million per year...within a 9 month timeframe."*

- Foxboro has seen *"for a $15 million capital investment over $10 million per year in operations savings;"* this is a 15 month payback period. Their decision to chose HP 3000s was based on the following criteria:

    - Proven technical innovations over several years.
    - Open architecture based on industry standards.
    - Scalable solutions from the desktop to the data center over an integrated operating environment.
    - Resources of a global supplier, with good local support and adaptation.
    - Broad third-party software solutions.
    - Excellent price/performance based on total cost-of-ownership, including purchase price, training, maintenance, supplies, software expense, power, air conditioning, floor space, and support staff.
    - Good purchase agreements and competitive terms.

- $1 billion Agorpur/Natrel *"...faced sales and billing problems that its mainframes couldn't solve. Four HP 9000s later, the company is saving $1 million per year."*

- Acustar (subsidiary of Chrysler), *"...better productivity, price/performance and lower maintenance cost."*

- Macleod-Stedman saw *"...$400,000 annual savings in system software costs ... $300,000 annual savings in staff costs ... $500,000 annual savings in other areas, including hardware maintenance and support."*

Providing Greater Value versus Mainframes Today ...

"On-time shipment and a 75% reduction in order cycle time."

- Foxboro

"70% reduction in staff cost."

- General Chemical

"50% reduction in MIS expenses without compromising services."

- Excellon

•
•
•

More than ROI

As discussed earlier, the benefits of user-based computing versus mainframe systems goes beyond a measurable return on investment.

- Corporations like Foxboro have seen how reshaping the business processes to meet changing and challenging environments has tremendous benefits *"...paperless, streamlined order flow...order entry manpower cut by 50%...on-time shipments and a 75% reduction in order cycle time...up to an 80-fold increase in engineering productivity..."*

- Again Macleod-Stedman benefited from business processes being reshaped *"...access to information measured in minutes instead of days ... substantial reduction in inefficient manual operations."*

- Fuji Bank stated that HP delivered, *"...cost effective alternative to a mainframe while providing flexibility through open systems."*

Forrester, 1992 determined in a survey that the exodus off of mainframe-based to user-based computing is being led by departments that need timely data for decision making. When ask which applications were fleeing the mainframe environment, users cited sales and marketing, and executive information systems (EIS) most frequently. Comments include:

- Transportation company: *"We need to keep track of barges and the materials on them. When this was on the host, we were paper intensive and the information was two days old. Now, users manipulate data into color graphics to determine where the barge is and what it is carrying."*

- Scientific equipment company: *"Our interest in 'downsizing' is increasing because our sales and distribution organizations want convenient access to data to develop their own reports."*

Currid & Company provides a summary of the mainframe alternatives benefits (84 respondents):

- *70% of sites achieved easier access to data,*
- *63% achieved improved flow of information,*
- *60% achieved significant cost reduction, and*
- *56% achieved greater flexibility in developing applications.*

At HP, we believe our current wins far exceed even these dramatic results. For more information on how you and your organization can benefit contact your local HP Sales Representative.

Don't Put Your Career on Hold

John Pickett

Hewlett-Packard
19490 Homestead Road, mailstop 43L8
Cupertino, CA 95014
408/447-2906

ABSTRACT:
For many companies, it is a requirement to integrate their telephone system with computer applications - is yours far behind? It is a fact that more companies than ever are using applications to control their telephone environment. These applications allow people in your company to sit at their desk, and place a phone call from their computer. Information on the caller can also be displayed, as the phone is still ringing.

This technology is upon us and is changing the way business is being done. This trend is inevitable and the choices are clear. To stay afloat in this ever changing world, learning about the integration of computers and telephone switches is a must.

Few thing have changed the computing industry like the PC revolution we all experienced in the early 80s. The growth of the PC and how it has affected our day to day business is something that few of us expected. This paper will describe another revolution that is current underway that will affect the way business is done for all of us - this revolution is the one that integrates our computing environment with our telephone systems.

Over the past decade, most of us have seen how we can integrate our telecom and computing by using the same transmission method. T1 lines gave us the capability to send a phone call and data traffic over the same wire. But the revolution I am referring to is the one that is taking place with your applications and how your company will be doing business now and in the years to come.

Why should you care about telecom and computer integration?

There are three main reasons you should be interested in this technology and how it gets integrated into your company.

1) This industry is going through explosive growth. The industry watchers that follow this integration foresee a 100% jump in growth for each of the next 3-4 years.
2) It solves REAL business problems. As new and exciting as this technology is, companies are not using applications that integrate their computers and telephone system at an expanding rate because it is new technology, they are doing so because it solves real business needs. The people who are making the decision on installing applications to work with your telephone switch are not the telecom managers or the MIS managers. The people who are making the decision to install this technology are the functional managers - the CEOs, the CFOs of companies.
3) Your job may depend upon it. As mentioned above, the people who are making the decisions to implement this technology are the functional level managers; as a result, the MIS staff is expected to fulfill the needs of the users and the choices are clear:
   A) You can ignore it and pretend that this trend in the industry is a short term fad which will never affect your company.
   B) You can pro-actively determine how this technology fits into your organization and be a hero for thinking of the idea first.

   We have seen people handle the situation in both ways. In the reactive mode - the person feels out of control for being asked to implement something they know nothing about. In the pro-active mode, this person has been perceived as a forward thinking individual looking out for the good of the company. Any guesses as to which one gets the promotion?

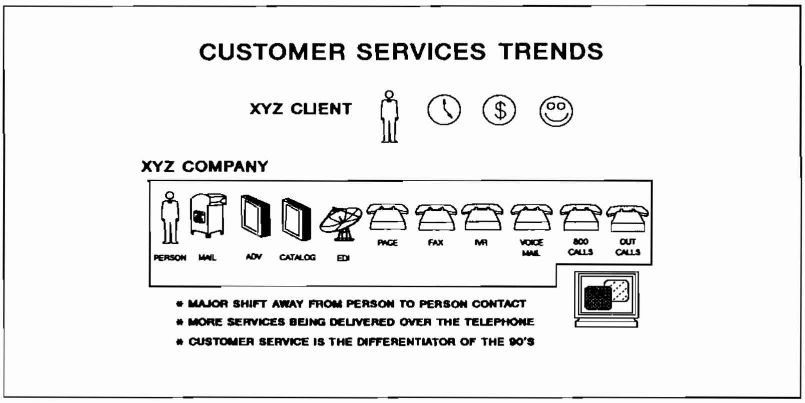
What are these REAL business problems?

Most of you have customers that you must communicate with, whether that is to sell products/services, support products/services, answer customer inquiries or perform on-going business communications.

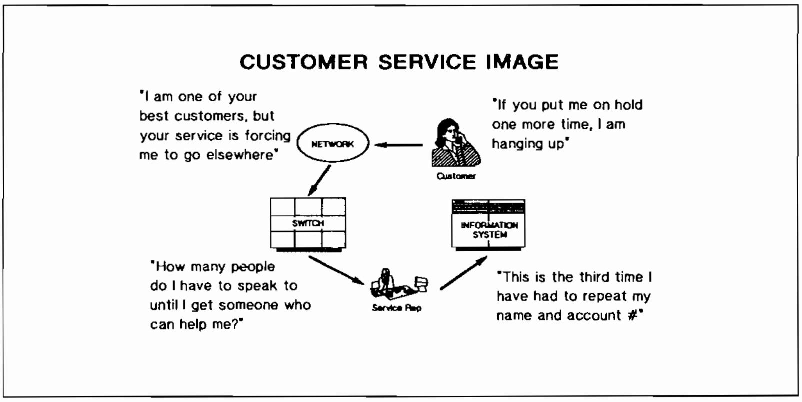
Two major trends are taking place today. The first is that more business communications are being performed over the phone. This is being driven by the rising cost of performing face to face contact. Two very good examples of this are the increased uses of telemarketing and tele-services like HP's tele-sales centers and response centers. Since many organizations are looking at lowering their operating costs, these expensive face to face sales visits are becoming less and less frequent.

This move away from face to face contact bring up the other problem many companies are facing. There is a trend towards "customer service" being viewed as a major competitive differentiator. Customers want more timely, cost effective and easy to use services from the vendors/service providers. This will become an even more competitive issue in the 90's.

The reasons for increasing customer service also have economic implications. It is 10 times more expensive to obtain a new customer than it is to maintain an existing customer.


CUSTOMER SERVICES TRENDS
XYZ CLIENT
XYZ COMPANY
PERSON
MAIL
ADV
CATALOG
EDI
PAGE
FAX
IVR
VOICE MAIL
800 CALLS
OUT CALLS
* MAJOR SHIFT AWAY FROM PERSON TO PERSON CONTACT
* MORE SERVICES BEING DELIVERED OVER THE TELEPHONE
* CUSTOMER SERVICE IS THE DIFFERENTIATOR OF THE 90'S

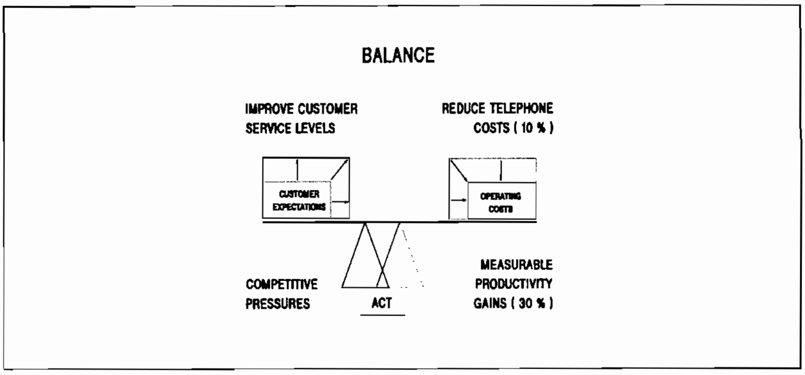
As companies move more towards telephone customer contact a new set of issues begins to emerge. Customers can begin to feel as though they are receiving less personal service. They can be put on hold too long, get transferred from agent to agent, department to department and at each step along the way, have to repeat information over and over again. Unless managed well telephone contact can result in a very frustrating experience for the end customer.


CUSTOMER SERVICE IMAGE
"I am one of your best customers, but your service is forcing me to go elsewhere"
NETWORK
Customer
"If you put me on hold one more time, I am hanging up"
SWITCH
INFORMATION SYSTEM
"How many people do I have to speak to until I get someone who can help me?"
Service Rep
"This is the third time I have had to repeat my name and account #"

The company's image that is being projected via "customer telephone contact" can be a competitive advantage if it is a positive experience or it can be a competitive disadvantage if it is a negative experience. With this shift toward more business being done over the phone, there have been complaints about people being on hold too long or being transferred to 3 or 4 people at the same company and having to repeat the same information to each person.

What has motivated other companies to integrated their telephone switch with their computers?

The dilemma companies are faced with is how they can improve their "customer service" image when at the same time they are being pressured to reduce costs and run a more cost effective operation.



BALANCE
IMPROVE CUSTOMER SERVICE LEVELS
REDUCE TELEPHONE COSTS (10 %)
CUSTOMER EXPECTATIONS
OPERATING COSTS
COMPETITIVE PRESSURES
ACT
MEASURABLE PRODUCTIVITY GAINS (30 %)

Customer expectations - competitive pressures are forcing companies to improve customer service levels to attract new customers or to keep the customers they already have. The interesting thing about customer service expectations is that they never decrease. Your customers don't understand receiving poor service just because your operating costs have decreased by 10%. Once a certain level of customer support has been reached, any decline in that level of service is perceived as poor customer service ... even it is better than what they can get from the competition.

Operating costs - competitive pressures are also forcing companies to reduce operating costs in order to provide cost competitive products / services. The major cost of a call center is composed of personnel costs and telecommunication costs. On the average the personnel cost is about 40% of the call center's budget and the telecom cost is another 40%.

Balance - There is a way of addressing this dilemma. You can reduce your telephone bill, improve your productivity and at the same time improve customer service levels. Many companies are very interested in finding out how.

What customers have this problem?

We tend to think that the large telemarketing firms are the only ones with this business problem. Not true, just think of how your company conducts business over the phone. How many of you have 800 numbers? How many of you provide customer assistance over the phone? Let's take a few industries and the types of applications that may be used:

Manufacturing industry -	
	Customer relations
	Product support
	Tele-sales
	Telemarketing
	Customer surveys
	Dealer Contact & Support
	...
Banking -	
	Customer relations
	Account inquiries
	Phone transactions
	Trading
	Tele-sales & marketing
	Collections
	...
Insurance -	
	Customer relations
	Policy inquires
	Policy changes
	Claims processing
	Claim's follow up
	...
Distribution -	
	Catalog fulfillment
	Order processing
	Order inquiries
	...

Publishing / newspapers ...
Service / travel / real estate ...
...

As you can see, many organizations have this problem. I'll bet that most of you have at least one, and probably more than one, internal operation doing a similar type of function.

How do you solve these business problems?

The solution to the business problem is to provide the most efficient management of customer calls and at the same time have your customers feel as though they are getting better service then they have been getting or can get else where.

Solution description

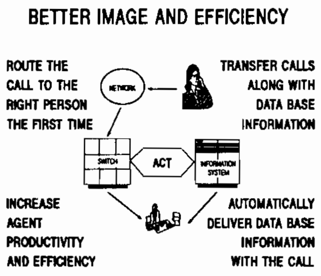
There are four ways of accomplishing this.

1) Route the customer call to the right person the first time. By using the phone number of the originating caller, we can route calls based on data base information, like which agent is best trained to answer a particular question or who is currently handling this customer.

2) Automatically provide, on the agent display screen, the customer's data base information as the call is being answered, thus eliminating unnecessary questions and time consuming data base retrieval steps.

3) If the customer needs to be transferred do not have them quizzed all over again. Automatically transfer screen information along with the call, thus eliminating unnecessary questions and even more data base retrieval time.

4) Let the computer do the laborious tasks, instead of the agents, like dialing the number, listening to the ring no answers and busy signals, allowing the agents to do more productive work.


BETTER IMAGE AND EFFICIENCY
ROUTE THE CALL TO THE RIGHT PERSON THE FIRST TIME
NETWORK
TRANSFER CALLS ALONG WITH DATA BASE INFORMATION
SWITCH
ACT
INFORMATION SYSTEM
INCREASE AGENT PRODUCTIVITY AND EFFICIENCY
AUTOMATICALLY DELIVER DATA BASE INFORMATION WITH THE CALL

The basic concept is to have your computer work in conjunction with the telephone system to provide a more efficient and intelligent operation.

In some environments an entire clerical staff is in place only to screen and route calls. This can be eliminated or drastically reduced.

Calls can be handled more efficiently resulting in a significant reduction in call duration. The reduction in call duration also affects the telephone bill. Because the call is shortened by about 30 seconds the bill is also shortened. Again bottom line cost savings!

A reduction in the call duration equates to better productivity. The average call can be reduced by about 30 seconds. This means that you can handle more calls with the same number of people or the same volume of calls with less people.

Bottom line savings!

How do you reduce operating cost?

For a typical call center handling phone calls for your company, there are 3 components:

1) Staffing the people required to answer the phones - included in this operating expense is salary, benefits and overhead. This accounts for 40% of the total cost for that call center.
2) Telecom cost - This includes the cost for the phone lines and long distance charges for the phone calls that are handles. This is another 40% of the total cost for that call center.
3) Equipment such as the PBX, computer and other hardware to assist the individual agent is the remaining 20%.

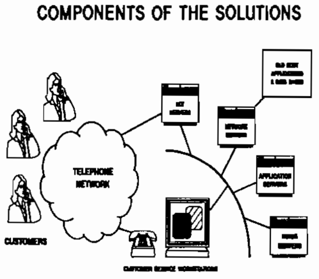
So if you were asked to cut your operating expenses, the obvious places to look are at the telecom and staffing cost. Great - but how do you do that?

For companies using this type of solution, they have found there is typically a time savings of 30 - 45 seconds for each phone call that displays profile information on the screen when the originator calls in. If you take this 30 seconds saving and multiply it across the number of phone calls a group within your organization receives over an 800 phone line, you can easily calculate the bottom line of your telecom savings. For those who are familiar with telecom cost, this is a very significant savings.

In addition, these comapnies have found that since each phone call is shorter in length , they can now handle more phone calls with the same number of agents. What this means to the managers of these call centers is they can control their cost much more effectively than they ever could have done in the past.

For a more detailed look at potential cost savings and revenue generating potential using this type of solution, contact your local HP representative.

What does this solution consist of?



A key component to the overall solution is the telephone network. Now don't tune out, you don't have to be a telephony expert, but you do need to understand some basics. The client will be calling in through the public telephone network and routed to either a PBX (private telephone switch) or a CENTREX service (a telephone company's business phone service). This will probably be an 800 number service and the telephone switch will have a function called ACD (Automatic Call Distribution) to queue the call and deliver it to the next available agent. We have all called a company and heard " All of our agents are busy, please wait and your call will be handled by the next available agent." Well this is an ACD function allowing more calls to be handled by a smaller number of agents.

The next component is the "glue" that brings together the telephony switch and the computer system together. For Hewlett-Packard, these products are called Applied Computerized Telephony (ACT). ACT works in conjunction with the telephone switch and the ACD to deliver to the computer application the client's telephone number and the number the customer called. This can be used by the application to infer who the customer is and why they are calling. The application can then direct the telephone switch to route the call to the appropriate agent and at the same time display the corresponding information on the agent's screen.

The next component is the application itself. The application can be an existing application, a new application yet to be written or a third party application package. ACT provides an easy to use API (application programming interface) that can be used by the programmer to integrate telephony functions into the application. Think of it as a tool kit.

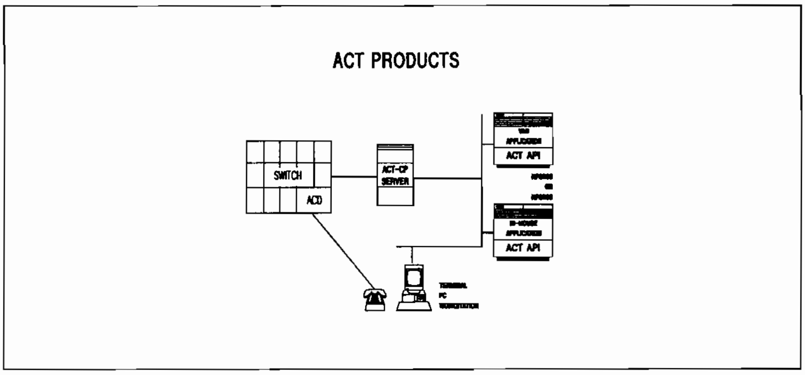
In many but not all environments, an "OLD" host application / data base needs to be integrated into the overall solution. This can best be accomplished by using a network server that takes 3270 screen information and displays it in an X-Window on the agents workstation or X-terminal.

Other servers may be part of the overall solution including image servers like HP's AIMS product.

The workstation can be a terminal, a PC or a UNIX workstation. This will depend on how the application has been written. For customers that have large IBM host data bases the workstation solution provides the best integration solution because the 3270 screens can be integrated into X-windows nicely.

Remember - the "glue" that integrates the telephone switch and teh computer systems is not the total solution. It is an enabling technology and a set of telephony integration products, like HP's ACT, allow you to offer a more complete solution to your own business problems.

What products provide this solution?



When using HP's ACT Call Processing two products are required, an ACT-CP Server and an ACT-CP API.

The ACT Call Processing Server is a pre-configured server bundle (hardware and software) that is customer installable. It is based on an HPUX processor although the customer does not need to know HPUX at all.

The ACT - Call Processing servers can work with the Northern Telecom PBX or Northern Telecom Central Office switch. Later on this year connection to another major U.S. PBX vendor.

The ACT Call Processing API's (Application Programming Interfaces) are available for both the HP3000 and HP9000 computer lines. The same API supports all ACT-CP servers.

The ACT Server communicates with the API's over a thin LAN connection therefore you will need to insure that the HP3000 and /or HP9000 has a LAN link connection. Because ACT utilizes standard TCP/IP sockets / NETIPC upper level networking services are not required.

Sounds Great - What do I do now?

The companies that have integrated their telephony environment with their computers have taken the following steps :

1) Qualify

If you look into your company and can answer yes to 5 or more of the following questions, then this type of solution has a very high probablity of helping you and your customers:

A) Is customer service a differentiator for your company?
B) Do you use an 800 service?
C) Do you have a PBX?
D) Do you use ACD software on your PBX?
E) Do you have people who answer the phone and use a computer to retreive customer information?
F) Do you service customers over the phone?
G) Would you like to automatically route important customers to special people inside your organization?
H) Does your company receive phone calls for multiple products or services?
I) Do you often transfer people when they call in?
J) Would you like to have an application automatically call people ? (Either a call back or a customer list)

2) Assessment

After you have determined that an integration of your environment would control operating cost by reducing your telecommunication or staffing cost, then you need to look at how the solution will fit into your environment. Each of the components mentioned above needs to have the necessary pieces so that they work together. You can choose to do it yourself, have an HP Consultant deliver a detailed assessment of your specific environment, or have it done by a third party system integrator.

3) Project management

Once you have determined what each component consists of, you need someone to coordinate the delivery of the necessary components. This person needs to work with the telephone switch vendor, public network provider, and HP computer personnel to coordinate a smooth installation. Once again your choices are to do it yourself, have HP deliver the necessary service, or have it done by a third party system intergator. At the completion of the project management phase, the infrastrucure to support an application exists, but the application still needs to be added.

4) Application assistance

The choices for applications can either be a VAB application that has already integrated the APIs in their application or to write these APIs into your own application - new or existing. Application assistance can be obtained from an experienced HP consultant or through a third party system integrator who has experience in integrating applications.

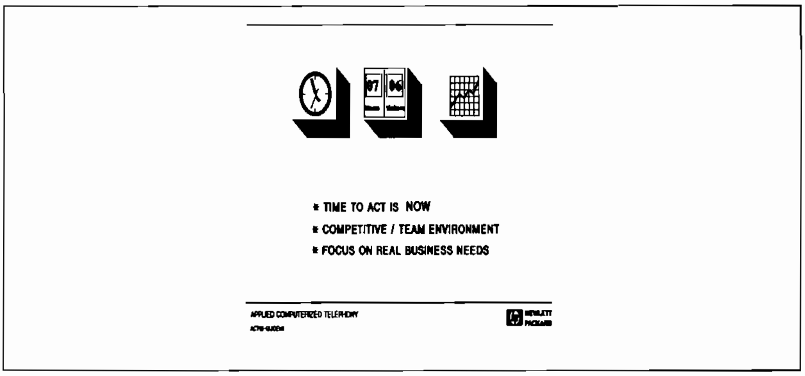
What have you learned ?

If there is anything you learn from this paper, it must be that the technology to integrate your phone system and computer applications is here - NOW. Companies are installing this technology to solve REAL business problems - controlling operating cost and especially improving their customer service.

If you are not actively investigating the integration of your telephone switch with your telephone, chances are your competition is.

The final question you must ask yourself is:

Can you afford not to ACT?



* TIME TO ACT IS NOW
* COMPETITIVE / TEAM ENVIRONMENT
* FOCUS ON REAL BUSINESS NEEDS

APPLIED COMPUTERIZED TELEPHONY
[illegible]

HEWLETT PACKARD

Paper #3863

HP X.25 Access Products : Providing Open Systems Solutions

by

Jean-Claude Viollier
Hewlett-Packard
Grenoble Networks Division
5, avenue R. Chanas
38053 Grenoble Cedex 09 - France

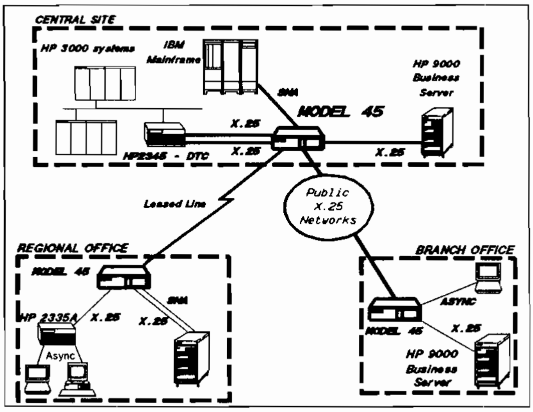
1. X.25 PACKET SWITCHING TODAY

The X.25 packet switching technology is today less and less viewed as a **"backbone network"** technology but more and more as a way to **"access"** either existing large corporate X.25 backbone networks or new backbone networks based on Routers, Frame-Relay or ISDN.

In order to provide a performant access to these Backbone Networks, HP has developed a complete family of cost-effective and performant X.25 Access products. The HP X.25 Access Products family gives you **a complete open system networking solution.**

2. REMOTE USER ACCESS TO CENTRAL HP 3000/9000s

The HP X.25 Access Products includes cost-effective, entry-level switching products such as the HP 2335A Asynchronous PAD and the HP Model 45 PAD/X.25 Multiprotocol Concentrator which are used to concentrate remote/distributed site access to HP 3000s or HP 9000 Business Servers central site.



This X.25 Access products allow

. Remote Terminal access to HP 3000/ HP 9000 Business Servers
. Concentration of multiple X.25 access lines in one location
. Integration of HP and IBM systems on one access point

The **major customer benefits** of these HP X.25 Access products are the following :

o Major savings on equipment costs and carrier bills

By concentrating over one or more physical lines the traffic coming from multiple HP systems and devices, the HP Model 45 X.25 Multiprotocol allows HP customers to save on equipment and line costs. For Public Data Network users, it allows concentrating multiple system and device access over one single line to the Carrier Network, thus saving on subscription costs.

o Excellent performance for transaction processing

Both 2335A and Model 45 are performant products that can typically fully use the physical line bandwith at speeds up 64 Kilobits-per-second. The HP Model 45 is a high performance X.25 Multiprotocol concentrator which is able to switch up to 1000 data packets per second.

o One single HP OpenView Network Management station to manage all HP Systems and Networking equipment (HP2335A, HP Model 45, HP Hubs, HP Bridges, Routers,...)

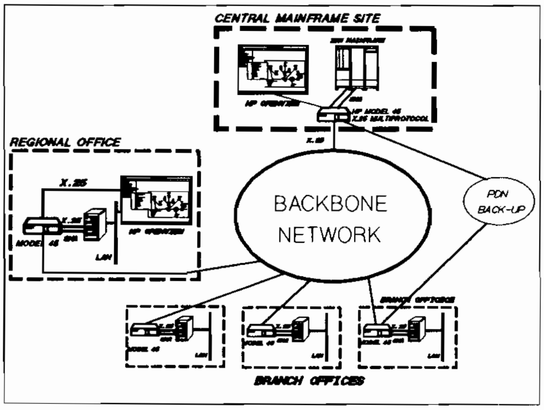
The HP OpenView Switch/PAD Manager which manages HP 2335A PADs and HP Model 45 Multiprotocol concentrators fully coexists on the same HP OpenView platform than other OpenView applications such as the HP OpenView DTC Manager. This allows HP customers to manage BOTH systems and their networks from ONE SINGLE HP OpenView station.

3. IBM MAINFRAME DOWNSIZING

Many HP customers are also IBM customers. This is the reason why HP offers IBM communication facilities on HP systems to connect them to IBM mainframes (SNA 3270, SNA 3770, LU 6.2, ...) over point-to-point dedicated leased lines using the IBM SNA/SDLC or IBM Bisynch communication protocols.

Among HP customers, many are buying HP 9000 Business Servers to run client/server applications, to send data to their central IBM mainframe - over one SINGLE communication infrastructure. Very often, this communication infrastructure is based on the X.25 technology and HP customers are looking for a low-cost "Multiprotocol Networking Access" product that allows their HP 9000 Business Servers to access their IBM host.

In this case, the HP Model 45 X.25 Multiprotocol concentrator is an excellent low-cost solution to allow this HP systems to communicate with the central IBM Mainframe over a private or public X.25 backbone network. Many large customers are using it both in US and Europe in this configuration.



In an SNA environment, the HP Model 45 provides the following features to HP customers :

- QLLC Protocol with Local Polling
- Support of HPAD/TPAD connections, NPSI/TPAD and HPAD/QLLC connections
- Support SNA Permanent Virtual Circuits (PVCs)
- Configurable SDLC frame size up to 4096 bytes
- Configurable SDLC station address
- Autocall on TPAD or HPAD with a Retry Timer
- Host-to-Host with NPAD mode
- Transparent Mode with XPAD mode.

PAPER #3864

Object-Oriented COBOL: The Next Generation

Megan Adams and Dmitry Lenkov

Hewlett-Packard Company
California Language Laboratory
11000 Wolfe Road, MS: 42U5
Cupertino, CA 95014

Abstract

Object-Oriented COBOL has excellent prospects for being the next generation of COBOL. It is a hybrid object oriented language which provides an evolutionary path to the benefits of object oriented technology for the community of COBOL developers. It is now being designed in the Object-Oriented COBOL Task Group, which has the charter to develop a specification which will be submitted for standardization.

Advances in research and implementation in the area of object oriented languages have enabled the design of a powerful feature set for Object-Oriented COBOL. (While most of the ideas presented here have been approved by the Object-Oriented COBOL Task Group, a few may be under discussion or not yet presented by Hewlett-Packard representatives in the Task Group.) We will focus on two important contributions made by Object-Oriented COBOL. The first is separating interface from implementation. The second focus of this paper will be the design of the object lifecycle, focusing in particular on the role of factory objects.

An earlier paper by the authors was published in the 1990 Interex Proceedings [3]. This paper provides an updated overview of the language features, and focuses in more depth on the particular contributions of Object-Oriented COBOL. It concludes with a report on the progress of the Object-Oriented COBOL Task Group.

1. Introduction

Object-Oriented COBOL addresses the classic software engineering problems of the growing complexity of large systems, the need to reuse and maintain existing code, the need to improve software productivity, and the desirability of incremental development of complex software. In addition, Object-Oriented COBOL is tailored to the particular requirements of the COBOL user community. This includes being able to handle complex systems consisting of old COBOL code.

Object-Oriented COBOL, an idea that originated at HP, is now undergoing definition through the Object-Oriented COBOL Task Group. In response to the HP initiative and to growing interest among COBOL vendors and in the COBOL user community, the development committee for ANSI and ISO COBOL standards (known as CODASYL) sponsored a symposium on Object Oriented COBOL in November of 1989. At this meeting the Object-Oriented COBOL Task Group (OOCTG) was formed, with the charter of drafting a specification for Object-Oriented COBOL.

Why Object-Oriented COBOL? In particular, why apply object orientation, an advanced technology, to COBOL, one of the oldest languages and a language some say is outdated? First and foremost, COBOL is still by far the most heavily used programming language. It has been recently estimated that approximately seventy billion lines of COBOL code are currently in use. Over two million COBOL programmers continue to maintain this existing COBOL code, as well as create new and more complex systems in COBOL [8].

Furthermore, it is important to note that the problems this large community faces are in fact the classic software engineering problems, faced by software developers everywhere. These include the growing complexity of large systems, the need to reuse and maintain existing code, the need to improve software productivity, and the desirability of incremental development of complex software [3]. Object-Oriented COBOL addresses all of these problems.

In addition, Object-Oriented COBOL will be tailored to the particular requirements of the COBOL user community. This includes being able to handle complex systems which may consist partly of old COBOL code. Object-Oriented COBOL provides support for embedding old COBOL programs within the Object-Oriented paradigm. Object-Oriented COBOL also provides COBOL programmers with a relatively easy transition to a new technology (while maintaining complete compatibility).

2. Object-Oriented Extensions to COBOL

Complexity is the single most important problem facing software developers [1],[2]. Object oriented programming is widely considered to be the most promising technique for successfully handling the growing complexity of software systems, as well as problems associated with this complexity, such as increased need for ease of maintenance and software reuse. While many parts of the software industry have to cope with this problem, the COBOL community might be the one most seriously affected by it. COBOL programmers have to deal with much larger and much older programs that continue to work and cannot be thrown away and rewritten.

Software complexity is an underlying problem which leads to other undesirable software traits such as reliability and maintenance problems. Complexity in software exists because of the growing complexity of the problems which developers are trying to address, but it is also usually magnified when abstracted through software. This additional complexity is incurred in part by the design methodologies which programming languages and tools support.

In addition, we believe object orientation is particularly well suited to data processing applications and COBOL. This is because object orientation provides a data centered approach, as it supports defining objects which consist of data and of operations on that data. Most COBOL applications are involved in some form of data processing, and their natural emphasis is on organizing and handling the data, rather than functional decomposition based on procedural logic.

What capabilities are needed to adequately address the problem of complexity?

- Object-Oriented COBOL should provide a sophisticated modeling capability. Normally in COBOL, a bank account problem is translated into a into a set of logical operations distributed among programs and files. Object-Oriented COBOL supports the approach of modeling the bank account objects and their associated operations directly in the Object-Oriented COBOL application.
- It is critical that maintainence be eased. According to some estimates, maintainence uses up to eighty per cent of software budgets. To address this problem, Object-Oriented COBOL must support modularity. Objects should be accessed through interfaces, so that changes behind the interface need not effect clients of the objects. For example, clients of bank account objects need not be effected by changes in bank account data layout or bank account service implementation, as long as these changes do not effect the interface to bank account objects. This approach eases maintenance and improves reliability.

– Software reuse must become easier. Reusability of software components is potentially the most important single factor in increasing software productivity. Object-Oriented COBOL supports this capability. For example, the definition of bank account objects can be re-used when constructing definitions of other classes of objects, such as checking account objects.

– We need to support software evolution. Applications need to be able to evolve to meet changing requirements. That is, existing definitions of classes of objects must be easily specialized and extended as new requirements arise. Object-Oriented COBOL supports this capability. The Object-Oriented COBOL class which defines bank account objects can be easily extensible to define credit card objects.

– Finally, we have to consider a large body of existing COBOL code (over 80 billion lines). Object-Oriented COBOL must be designed as an extension of the existing language rather than a new language. This means that every legal COBOL program must be legal in Object-Oriented COBOL. It also means that an existing COBOL program can be encapsulated by some of the new constructs to allow its reuse in a way consistent with the new approach introduced by Object-Oriented COBOL. It follows from these two requirements that the syntax of Object-Oriented COBOL should be a natural and consistent extension of existing COBOL syntax.

In the remainder of this paper, we will be looking in more detail at specific concepts and features of an Object-Oriented COBOL, with particular attention to how the problem of complexity is addressed. The focus will be on two important contributions made by Object-Oriented COBOL. The first is separating interface from implementation. That is, the inheritance hierarchy is separated from the interface hierarchy, where code reuse is supported by inheritance and the subtyping mechanism is based on interface conformance. This system provides the benefits of code reuse associated with inheritance as well as providing the benefits of interfaces as functional descriptions and boundaries of encapsulation. It also preserves the expressiveness that gives untyped languages their flexibility [6], while providing safe static typing. The second focus in this paper will be the design of the object lifecycle, focusing in particular on the role of factory objects.

3. Class Definition

Here is an example Object-Oriented COBOL class definition. The simple BANK-ACCOUNT class definition shown below defines the object data needed by account objects (such as BALANCE and ACC-NUMBER) and three services which operate on that data, DEPOSIT, WITHDRAW, and RETURN-BALANCE.

```
IDENTIFICATION DIVISION.
 OBJECT-CLASS. BANK-ACCOUNT. /* class definition.
 DATA DIVISION.             /* object variables declared here.
 WORKING-STORAGE SECTION.
 01   BALANCE                 PICTURE $$$,$$$.99 VALUE ZERO.
 01   ACC-NUMBER              PICTURE 9(14).
 01   CUST-INFO.
   05   CUST-NAME             PICTURE X(20).
   05   CUST-SS-NUM           PICTURE X(9).
 PROCEDURE DIVISION.                   /*** object methods declared here.

     IDENTIFICATION DIVISION.          /*** 1st method declaration.
     METHOD-ID. RETURN-BALANCE IS PUBLIC.
     DATA DIVISION.                    /*** method local data declared here.
     LINKAGE SECTION.                  /*** formal parms section.
     01   ACCT-NUM                PICTURE 9(14).     /*****parameter
     01   ACCT-BAL                PICTURE $$$,$$$.99./*****return value
     PROCEDURE DIVISION
       USING ACCT-NUM RETURNING ACCT-BAL./*** method prototype.
       MOVE ZEROS TO ACCT-BAL.         /*** method code begins.
       IF ACC-NUMBER = ACCT-NUM MOVE BALANCE TO ACCT-BAL.
       EXIT METHOD RETURNING ACCT-BAL.
     END METHOD RETURN-BALANCE.        /*** method code ends.

     IDENTIFICATION DIVISION.
     METHOD-ID. WITHDRAW IS PUBLIC.  /* 2nd method declaration.
     /* withdraw from account;  code not shown
     END METHOD WITHDRAW.

     IDENTIFICATION DIVISION.
     METHOD-ID. DEPOSIT IS PUBLIC.   /* 3rd method declaration.
     /* deposit into account;  code not shown
     END METHOD DEPOSIT.

     IDENTIFICATION DIVISION.
     METHOD-ID. PRINT IS PUBLIC.    /* 4th method declaration.
     /* print account info; code not shown
     END METHOD PRINT.
  END OBJECT-CLASS BANK-ACCOUNT.
```

Figure 1. ACCOUNT Class Definition

4. Inheritance

Inheritance is useful for extending the functionality of existing classes. For example, we can extend the the functionality of BANK-ACCOUNT so that it can handle credit cards. A new class, CREDIT-CARD, can be defined. This class will inherit (and thus reuse) the services and data of BANK-ACCOUNT and will add new services and data, to handle credit card functionality. Inheritance is specified in the INHERITS clause, as shown in the example. One of the inherited methods, RETURN-BALANCE, is re-implemented in the descendant class. The method parameters change as well as the method body. Note that in traditional object oriented languages, which base subtyping on inheritance, redefinition of parameters in polymorphic methods is not legal. Because Object-Oriented COBOL separates subtyping from inheritance, and bases subtyping on interfaces, this form of redefinition is legal. Note Object-Oriented COBOL does not allow overloading.

```
IDENTIFICATION DIVISION.
 OBJECT-CLASS. CREDIT-CARD INHERITS BANK-ACCOUNT /* inherits clause.
              RETURN-BALANCE IS NEW CODE        /* method re-implemented.
              PRINT          IS NEW CODE.       /* method re-implemented.
 DATA DIVISION.                 /*** object variables declared here.
 WORKING-STORAGE SECTION.
 01   CREDIT-LIMIT              PICTURE $$$,$$$.99 VALUE ZERO.

 PROCEDURE DIVISION.            /*** object methods declared here.
     IDENTIFICATION DIVISION.
     METHOD-ID. RETURN-BALANCE IS PUBLIC. /*** re-implemented method
     DATA DIVISION.
     LINKAGE SECTION.           /*** formal parameters
     01   ACCT-NUM                         PICTURE 9(14). /*** parameter
     01   CREDIT-INFO.          /*** return value; note type changed!
       05   ACCT-BAL                       PICTURE $$$,$$$.99.
       05   AVAIL-CREDIT                   PICTURE -$$$,$$$.99.
     PROCEDURE DIVISION                             /*** method prototype.
       USING ACCT-NUM  RETURNING CREDIT-INFO.
       MOVE ZEROS TO ACCT-BAL.                      /*** method code
       IF ACC-NUMBER = ACCT-NUM
          MOVE BALANCE TO ACCT-BAL
          COMPUTE AVAIL-CREDIT = CREDIT-LIMIT - BALANCE
       END-IF.
       EXIT PROGRAM RETURNING CREDIT-INFO.         /*** returns new type
     END METHOD RETURN-BALANCE.

     IDENTIFICATION DIVISION.
     METHOD-ID. PRINT IS PUBLIC.                   /*** re-implemented method
     /* print account info; code not shown
     END METHOD PRINT.
  END OBJECT-CLASS CREDIT-CARD.
```

Figure 2. CREDIT-CARD Class.

5. Interface Definition.

Classes and inheritance in Object-Oriented COBOL are primarily mechanisms for code re-use. Classes define implementation, and inheritance defines new implementations in terms of already existing implementations. Modularity and generic code are supported by interfaces, which can also be used to define subtyping relationships between objects.

An interface contains only what a client needs to know in order to use the class. It does not expose implementation or data representation. An interface is a list of services available from objects of certain classes. Data definitions are not included in the interface.

```
IDENTIFICATION DIVISION.
  OBJECT-INTERFACE. CHECKING-ACCOUNT.

   IDENTIFICATION DIVISION.
   METHOD-ID. DEPOSIT.
   LINKAGE SECTION.              /* formal parameter definitions
   01   TRANSACTION-AMOUNT           PICTURE $$$,$$$.99.
   01   NEW-BALANCE                  PICTURE $$$,$$$.99.
   PROCEDURE DIVISION USING TRANSACTION-AMOUNT RETURNING NEW-BALANCE.
   END METHOD DEPOSIT.

   IDENTIFICATION DIVISION.
   METHOD-ID. WITHDRAW.
   LINKAGE SECTION.              /* formal parameter definitions
   01   TRANSACTION-AMOUNT           PICTURE $$$,$$$.99.
   01   NEW-BALANCE                  PICTURE $$$,$$$.99.
   PROCEDURE DIVISION USING TRANSACTION-AMOUNT RETURNING NEW-BALANCE.
   END METHOD WITHDRAW.

   IDENTIFICATION DIVISION.
   METHOD-ID. PROCESS-CHECK.
   LINKAGE SECTION.              /* formal parameter definitions
     01 CHECK.
       05   CHK-AMOUNT                PICTURE $$$,$$$.99.
       05   CHK-DATE                  PICTURE 99/99/99.
       05   CHK-NUMBER                PICTURE 9(4).
       05   CHK-CUST-INFO             PICTURE X(29).
       05   CHK-ACC-NUM               PICTURE 9(14).
     01 NEW-BALANCE                PICTURE $$$,$$$.99.
   PROCEDURE DIVISION USING CHECK RETURNING NEW-BALANCE.
   END METHOD PROCESS-CHECK.

   IDENTIFICATION DIVISION.
   METHOD-ID. RETURN-BALANCE.
   LINKAGE SECTION.              /* formal parameter definitions
   01   ACCT-NUM                   PICTURE 9(14).
   01   ACCT-BAL                   PICTURE $$$,$$$.99.
   PROCEDURE DIVISION USING ACCT-NUM RETURNING ACCT-BAL.
   END METHOD RETURN-BALANCE.

   IDENTIFICATION DIVISION.
   METHOD-ID. PRINT.
   LINKAGE SECTION.
   01  ACCT-INFO        OBJECT REFERENCE TEXT-OBJ
   PROCEDURE DIVISION USING ACCT-INFO.
   END METHOD PRINT.

END OBJECT-INTERFACE CHECKING-ACCOUNT.
```

Figure 3. CHECKING-ACCOUNT Interface Definition.

Normally, a user can assume that an interface such as the one that appears in Figure 3 exists for each class, possibly created by the compiler. That is, a user can assume that for each class, an interface exists such that objects of the class can satisfy requests for any of the services described in that interface.

In Object-Oriented COBOL, object data is never part of the interface. Data is always accessed through procedures whose prototype can be included in the interface. Data and code are encapsulated behind the interface, contributing to modularity.

Interfaces do not need to be associated with implementations which are defined in particular classes. In this case, interfaces support what is known as generic code. This allows a client to execute services provided by any object which satisfies the interface and whose class provides an implementation of these services. For example, the bank account system includes ATM-TRANSACTION objects. These objects issue WITHDRAW and DEPOSIT requests, based on customer input. ATM-TRANSACTION objects do not need to know whether this account is a credit card account, or a savings or checking account. For these atm transactions, the designers could provide a GENERIC-ACCOUNT interface, which defines just the deposit and withdraw services.

6. Separation of Interface from Implementation.

In Object-Oriented COBOL inheritance is a mechanism for creating classes incrementally: one class, the descendant, can be defined in terms of other classes, called the ancestors. The descendant class extends the ancestor classes by possibly adding data to the object representation (data format), possibly adding new services, and possibly changing the definition of existing services. Inheritance is a mechanism for code reuse, and it also eases maintainence by making it easier to evolve existing code in accordance with changing specifications.

To fully support data abstraction, Object-Oriented COBOL, also defines interfaces as entities separate from objects and classes. Interfaces in Object-Oriented COBOL provide a classification of objects based on the services they provide to clients. An object satisfies an interface if it provides all the services defined in the interface. An object can satisfy several interfaces.

Interfaces are important both as a description of how an object is used and as a way of checking for certain erroneous uses of objects. These errors can often be detected at compile time, which leads to improved software quality and productivity, particularly necessary in the commercial environment. Only objects that satisfy a particular interface are valid providers of the services specified in that interface. Object-oriented code is written by assuming that various objects can handle corresponding sets of requests for services, i.e., in terms of interfaces of objects. Interfaces make it easier to reuse existing software and to evolve existing software to fit changing specifications.

Many object-oriented languages define rules whereby an object of one type can be used where an object of another type is expected. These rules define subtyping relationships, where the type of the first object is a subtype of type of the latter object. Many subtyping systems are based on inheritance. In these systems, if one class inherits from another, it is automatically defined to be a subtype of the parent class.

With inheritance-based subtyping, inheritance must be restricted to ensure that parent and child classes always have a subtype relationship. That means methods cannot be freely renamed, redefined, or deleted in child classes [5,6]. Also, clients of a class are exposed implicitly to the inheritance hierarchy when they assume a subtype relationship between related classes [4]. If the inheritance hierarchy changes, client code is broken. This is a serious violation of encapsulation, because the client has no way of knowing about the change and should not be effected.

One way of addressing this problem is to separate subtyping from inheritance [3,4,5,6]. This approach is adopted in Object-Oriented COBOL, where interface conformance is the basis of subtyping, and inheritance is simply a code re-use mechanism. One interface conforms to a second interface if it contains all the services the second interface contains (and possibly additional services). Objects are accessed through various interfaces. Interface conformance provides the basis for generic code, because it specifies when an object accessed through one interface can be used where an object accessed through another interface is expected. This mechanism turns out to be more flexible than the subtyping mechanism based on inheritance, because the same interface can be supported across entirely unrelated class hierarchies. Thus, for example, objects of unrelated classes can support the same print interface.

7. Generic Code and Interfaces.

It is important to note that interfaces can be defined entirely separate from classes. While every class has an interface associated with it (which has the same name as the class), there can be interfaces which are not associated with any class. For example, we may want to define a PRINT-INTERFACE, which simply defines a PRINT method prototype (see Figure 4).

```
IDENTIFICATION DIVISION.
  OBJECT-INTERFACE. PRINT-INTERFACE.

  IDENTIFICATION DIVISION.
  METHOD-ID. PRINT.
  LINKAGE SECTION.
  01 PRINTABLE-OBJ OBJECT REFERENCE TEXT-OBJ.
  PROCEDURE DIVISION USING PRINTABLE-OBJ.
  END METHOD PRINT.

END OBJECT-INTERFACE PRINT-INTERFACE.
```

Figure 4. PRINT-OBJ Interface Definition.

With the example in Figure 4, if every printable object implements a PRINT service for its print functionality, then a program can access this service through the PRINT-INTERFACE to issue generic print invocations for all printable objects. (In a more detailed example, the invocation may also specify a device object where the document will be printed.) For example, all the previous examples included the PRINT service.

Generic code is a major factor in the reusability of object oriented programs. Generic code can be used for different purposes when the invocations are sent to objects that interpret them differently.

In our BANK-ACCOUNT example, the interface CHECKING-ACCOUNT contains all the services defined in the BANK-ACCOUNT interface, with a few added. This brings up the general point that one interface can conform to another interface. This means that the set of services defined in one interface includes the set of services defined in the other interface, and it is called interface conformance. Interface conformance defines a relationship between object interfaces.

Generic code relies on the fact that multiple (non-identical) interfaces can conform to the same interface. When one object has an interface which conforms to the interface of another object, then the first object can be used wherever the second can be used. In our example, because the interface for CHECKING-ACCOUNT conforms to the interface for BANK-ACCOUNT, an object with the CHECKING-ACCOUNT interface can be used interchangeably with an object with a BANK-ACCOUNT interface, wherever an object with the BANK-ACCOUNT interface is expected. That is, since the CHECKING-ACCOUNT interface includes (at least) all the services of BANK-ACCOUNT, an object of the CHECKING-ACCOUNT interface can satisfy any request made of a BANK-ACCOUNT object.

Conformance needs to be explicitly asserted. Both class and interface definitions can explicitly define conformance. In this example, the CHECKING-ACCOUNT class definition specifies an interface that contains the interface to BANK-ACCOUNT. That is, there is a conformance relationship between the CHECKING-ACCOUNT interface and the interface to BANK-ACCOUNT. This is explicitly asserted in the following manner:

```
IDENTIFICATION DIVISION.
OBJECT-CLASS. CHECKING-ACCOUNT INHERITS BANK-ACCOUNT
                               CONFORMING TO BANK-ACCOUNT.
```

Figure 5. Conformance Specification.

Now the interface common to both CHECKING-ACCOUNT and BANK-ACCOUNT defines generic code.

However, this conformance needs to be verified by the compiler. If it is found to be invalid, a compile-time type check error will be reported.

8. Factories.

We have so far just shown the object portion of the class definition, which is the portion of the class definition which defines data templates and method implementations for objects of the class. A class definition may have a factory portion as well as an object portion. Just as the object portion contains object data and object methods, the factory portion contains factory methods and factory data. Factory data, however, is different in one important respect from object data. Factory data is data that is shared for all instances of a class (and factory methods operate on this data). This contrasts to the data definitions in the object portion of the class definition, which provide a common data layout for all objects of the class, but the memory allocated for the data is unique to each object.

If the factory portion is not present in the class definition the compiler will generate a default factory object with default factory method implementations. The factory object must be generated because the factory object is responsible for creating new instances of the class.

A simplified syntactic skeleton for a class definition is shown below, with the factory portion included.

```
IDENTIFICATION DIVISION.
OBJECT-CLASS. <object-class-name>. /* class definition

[FACTORY.                          /* factory definition
[DATA DIVISION.                    /* factory data
  [Factory data]]
[PROCEDURE DIVISION.
  [Factory methods]]               /* factory methods
END FACTORY.]

[OBJECT.                           /* object definition
[DATA DIVISION.                    /* object definition
  [Object data]]                   /* object data
[PROCEDURE DIVISION.
  [Object methods]]                /* object methods
END OBJECT-CLASS <object-class-name>.
```

Figure 6. Class Definition Structure.

The factory portion of a class definition is compiled to a single factory object. The diagram below shows that compiling a class definition produces three things: a set of compiled methods that constitute the shared code of the object instances, a factory object, as well as an interface (which some implementations may produce) that all objects of this class will satisfy.

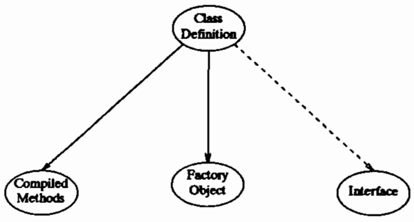


Figure 7. Compiling a Class Definition Produces Three Things.

The factory object is responsible for creating and destroying new object instances. Since it must be available to perform these tasks at the start of execution, the factory object is allocated and initialized automatically at the start of the execution of the run-unit. It is never allocated by the user. There is only one factory object per class, and the factory data is in this object, which allows all the objects created by the factory to share the data in the factory. Factory methods can be freely redefined in descendants.

9. The Object Lifecycle.

The various stages in the life of an object (creation, initialization, use, finalization, and destruction) are collectively referred to as the object lifecycle. The factory is an integral part of the object lifecycle because factories are responsible for the creation of new objects. The factory can also define the implementation of object destruction. The factory carries out its tasks of creating and destroying new objects via the factory methods CREATE and DESTROY. These methods are always present in the factory. Defaults are generated by the compiler which can be overridden by user coded versions. CREATE is a private factory method which is invoked only by the COBOL statement INITIALIZE, as shown below.

```
INITIALIZE BANK-ACCOUNT RETURNING CUST-ACCOUNT.
```

The factory object always has the class name, so in effect the INITIALIZE statement above is making a request of the factory (BANK-ACCOUNT) to make a new account. The INITIALIZE statement passes a parameter to CREATE indicating how much memory to allocate; users do not need to specify this value.

DESTROY is also a private factory method. It cannot be invoked in response to any user action. It is not available through the INITIALIZE statement, and it is private, which means it is not available to any client of the class. In general, objects are only destroyed when the automatic memory management system determines that they are no longer referenced. This is much safer than having user controlled memory management.

Like CREATE and DESTROY, initialization and finalization can be specified in user defined methods. There are object initialize and finalize methods, declared in the object portion of the class definition, and factory initialize and finalize methods, declared in the factory. If the user does not specify these methods, the compiler generates defaults.

Object lifecycle method characteristics are summarized in the following table.

Lifecycle Method Characteristics		
	Visibility	*Type*
Create	Private	Factory
Destroy	Private	Factory
Initialize Code	Private	Instance
Initialize Code	Private	Factory
Finalize	Private	Instance
Finalize	Private	Factory

The fact that all lifecycle methods are private, invoked only through the statement INITIALIZE, helps to insure that objects are in a consistent state. For example, the fact that only INITIALIZE (not users) can invoke object initialize code on dynamically allocated objects helps to insure that no such objects will exist which have been created but not initialized. Similiarly, the fact that users cannot invoke the object FINALIZE method helps to insure that no objects in the system are finalized but not destroyed. This results in a system whose behavior is more consistent.

Other goals that were considered when this model of the object lifecycle was designed were minimizing the impact of lifecycle methods on interface conformance, and maximizing the robustness of the resulting applications. The latter was accomplished by requiring automatic memory management, which is much safer than allowing users to destroy existing objects. The former was accomplished by making all lifecycle methods private, so they do not effect any interfaces, and thus have no impact on interface conformance relationships. In addition, the user interface to the object lifecycle consists of the single statement INITIALIZE, which is simple to understand and use.

10. Object-Oriented COBOL Task Group progress report

In November of 1989, in response to an HP initiative and to growing interest among COBOL vendors and in the COBOL user community, the CODASYL COBOL Committee sponsored a symposium on Object-Oriented COBOL. At this meeting the Object-Oriented COBOL Task Group (OOCTG) was formed, with the charter of drafting a specification for Object-Oriented COBOL. The OOCTG has been meeting quarterly since January 1990, and its participants include both vendors and users (such as IBM, HP, Hitachi, Realia, Micro Focus, Digital, Sun, Southwestern Bell, & Hewitt Associates).

At its December 1991 meeting, the OOCTG selected a simple and consistent subset of features from all those which have been investigated for inclusion in Object-Oriented COBOL, and called that subset Version One. Version One was presented to the OOCTG parent committees (ANSI X3J4 COBOL Committee and the Codasyl COBOL Committee) on January 16, 1992. Version One represents an important milestone in that it includes powerful functionality and provides those interested in Object-Oriented COBOL with an understanding of the OOCTG technical approach. (For an overview of the features included in Version One see [7].) It is also possible that some of the early implementations of Object-Oriented COBOL will use Version One as a guide.

The final Technical Report submitted by the OOCTG to its parent committees will include some additional features that are not included in Version One, in particular, exception handling. The schedule for submission of this Technical Report is aggressive.

References

1. G. Booch, *Object Oriented Design With Applications* , Benjamin/Cummings Publishing Company, Inc. Redwood City, California, 1991.

2. D. Lenkov, M. Beckmann, *C++: A New Software Development Methodology,* Proceedings of the Interex HP Users Conference, Boston 1990, Volume 1, 1990.

3. M. Adams, D. Lenkov, *Object Oriented COBOL: A New Initiative From Hewlett-Packard,* Proceedings of the Interex HP Users Conference, Boston 1990, Volume 1, 1990.

4. A. Snyder, *Inheritance and the Development of Encapsulated Software Systems,* Research Directions in Object-Oriented Programming, B. Shriver & P. Wegner, editors, The MIT Press, Cambridge, Massachusetts, 1987.

5. William Cook, Walter Hill, Peter Canning, *Inheritance is Not Subtyping,* Proceedings of the ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL) January 1990.

6. P. Canning, W. Cook, W. Hill, W. Olthoff, *Interfaces for Strongly-Typed Object-Oriented Programming,* OOPSLA Conference Proceedings, Volume 24, Number 10, ACM Press, 1989.

7. M. Adams, *Report on the Object-Oriented COBOL Task Group,* Hotline on Object-Oriented Technology, Vol. 3, No. 5, March 1992.

8. Ed Yourdon, *Report on OOPSLA'89,* American Programmer, Ed Yourdon's Software Journal, Vol. 2, No. 11, November 1989.

11. Appendix: Terminology and Concepts

The terminology and concepts defined below have been revised since their definitions appeared in our previous Interex paper [3], and are therefore included below.

Class

A class (or object class) defines the implementation of a data abstraction, including the format of the data associated with an object and how the services provided by the class manipulate that data. Different objects, by being of the same class, share a common implementation. Objects of related classes can also share implementation, if the classes that define them have an inheritance relationship. (Inheritance is a mechanism for defining an object class in terms of other classes; see Class Inheritance.)

In general, classes use object data and factory data to define the representation of state information, and object methods and factory methods to implement requests for services. Although the executable code is shared among objects, each object typically has its own internal state. The creation of new objects is called instantiation.

Class Inheritance

Class inheritance (called inheritance here for convenience) is a mechanism for creating classes incrementally: one class, the descendant, can be defined in terms of other classes, called the ancestors. The descendant class is built upon the ancestor classes by possibly adding data to the object representation (data format), possibly adding new operations, and possibly changing the definition of existing operations. Single inheritance permits an class to be defined only in terms of a single ancestor class. Multiple inheritance allows more than one ancestor class to be used in defining a descendant class.

Conformance

Conformance in Object-Oriented COBOL is a classification of interfaces (and therefore objects) in terms of their services. One interface conforms to another if it contains the set of requests in the other interface. This conformance relationship between interfaces establishes that an object of the first interface can be used wherever an object of the second interface is expected. This is because an object of the first interface will be able to respond to any request made of an object of the second interface (and maybe more). Conformance allows Object-Oriented COBOL applications to be statically type checked, so that at runtime objects are not requested to perform services that they do not support.

Dynamic Binding

Binding is the selection of a method to provide a service. Binding is dynamic when the selection of a method to provide a service is made at the time the service is actually invoked–that is, at application runtime. Dynamic binding is often associated with polymorphic invocations.

Encapsulation

Encapsulation is the hiding of implementation (code and data) behind an interface; the interface is the "boundary" of encapsulation. The advantage of encapsulation is that changes of implementation that do not change the interface do not effect users of that interface. Encapsulation can also be understood as enforced data abstraction.

Embedded Programs

An COBOL program that is located within an class definition is called an embedded program.

Invocation

An invocation is a statement that specifies a service to be carried out by an object. An invocation specifies a service and an object that is to provide the service. This object is the provider of the service; it is called the provider object. An invocation may take arguments and produce results.

Object

An object (or object instance) is a computational entity whose behavior (the ability to access or modify the information associated with it) is characterized by a set of services that it can perform in response to requests from clients (the programs or other object instances making the requests). The services are described without dependencies on the particular data structure or data format used to represent the information or the algorithms used to implement the operations. In particular, there could be several possible implementations of the same behavior. In object oriented programming, services are often called operations or methods.

Three primary uses of objects can be identified. Some objects model things in the domain of an application, such as cities and roads in a program for making maps. Other objects provide an interface to resources in a computer system, such as modems in a computer network. And others model mathematical abstractions, such as stacks and arrays, which are used in programming.

Object Lifecycle

An object lifecycle consists of the various stages in the life of an object. It comprises object creation, initialization, use, finalization, and destruction (deallocation).

Object Lifecycle Methods

These methods are pre-defined and handle various aspects of the object lifecycle, including object creation, initialization, finalization, and destruction (deallocation).

Object Interface

An object interface is a named set of services that provide access to an object. This interface need not include all services defined by the object class.

Object Reference

An object reference is a unique identifier for the object. An object reference provides a way of unambiguously identifying an object. An object reference might take the form of a name or a location.

Polymorphic Invocation

A polymorphic invocation is an invocation that may be issued to different objects that provide (similar) services with different implementations and possibly different behaviors. The polymorphic invocation itself does not determine how the services will be performed. When a polymorphic invocation is issued, a selection process determines the actual code to be executed to perform the service.

A polymorphic invocation is sometimes called a generic request. Polymorphic invocations support generic code.

Static Binding

Binding is the selection of a method to provide a service. Binding is static when the selection of a method to provide a service is made before the service is actually invoked. Static binding is often associated with non-polymorphic invocations.

Paper #: 3865

New Directions for COBOL ... Where will COBOL 2000 Take Us

Julia Rodriguez
Hewlett-Packard
Mail Stop 42U5
11000 Wolfe Road
Cupertino, CA 95014-9804
juliar@cup.hp.com

COBOL is not only the language of yesterday, and today. It will be the language of tomorrow, as well. The next standard may take COBOL to the cutting edge of language development. Where will that cutting edge lie?

The X3J4 COBOL committee hereafter referred to as X3J4 is responsible for the development and interpretation of the ANSI and ISO COBOL standard. It is the standards committee's responsibility to see that the COBOL standard is both timely and necessary. X3J4 is currently accessing the needs of the COBOL community to project what will be needed in the next COBOL standard. Referring to the next standard as COBOL 2000 is perhaps a bit facetious, but at this writing there is no firm schedule for the next standard.

This paper will discuss changes to the standard since 1985. It will look at the process being developed for the next standard. Next it will discuss features that are being considered for the next full revision of the standard. Finally, it will appeal to the COBOL community to get involved in standardization.

1 What is the current state of the standard?

The last full revision of the standard took place in 1985, known as ANSI COBOL X3.23-1985. Its major contribution (in addition to many small improvements and corrections) was adding structured programming features to the language. This standard was eleven years in the making. X3J4 next attempted to speed up standardization by producing addenda, now called amendments.

Intrinsic Functions addendum was the first. The Intrinsic Functions addendum became an official part of the standard in 1989, officially known as ANSI X3.23a-1989. It added string, arithmetic, date, and business functions to the standard.

The purpose of the Corrections Amendment is to fix problems and ambiguities introduced by the 1985 standard and the Intrinsic Functions addendum. This is needed as no specification is entirely error free. X3J4 felt these changes were needed in a timely fashion to prevent program dependence on ambiguous specifications. The third public review for the draft Corrections Amendment has just ended. It is expected that this amendment will need no further changes and will become an official part of the COBOL standard specification in about 2 years. This amendment will have little impact on COBOL programmers, as most implementors have already implemented the changes specified in the Corrections Amendment.

2 Amendments versus Full Revision

The committee has tried a new method of standardization for the last 7 years. Instead of trying to revise the entire standard, the committee produced addenda or amendments to the standard. There is no difference between the terms amendment or addendum, terminology in the standards world changed between the Intrinsic Functions Addendum, and the Corrections amendment.

Amendments were an experiment in standardization. This experiment is at an end. Future work will probably be in the form of full revisions of the standard. Amendments had the following disadvantages: 1) an amendment takes up just as much administrative time for X3J4 as a full revision; 2) amendments are hard to read and understand because they are published as line changes to the main standard; 3) they have continued to be controversial because the language enhancements than lend themselves to amendments are limited. For the work involved, X3J4 feels that more substance should be produced and is leaning toward full revisions of the standard in future rather than producing smaller amendments.

3 Revision Plans

The X3J4 committee is trying to define the shape and form of the next revision of COBOL. The first schedules and predictions would have had a new COBOL standard in the year 2002. The committee is currently defining a "Plan B" for the next revision. This plan would have a new standard by the year 1997. In this plan X3J4 would produce a revision, in short order, by limiting the feature set to whatever can be completed by 1996. In parallel, work could continue on revision work for a later more complete revision.

4 Features being considered for the Next Full Revision

Described here are some of the interesting features being considered for the next full revision. X3J4 has attempted to group the features into one or more of the following categories.

4.1 Application User Interface

These items each add the ability for the programmer to specify a user interface for an application.

• Forms Interface Management System (FIMS). A FIMS standard will be official within a year. New verbs are being considered to access this subsystem. FIMS would allow portable application user interface. COBOL applications would no longer be dependent on the screen or window management of a particular vendor. However, adding new verbs to access this subsystem is controversial. Programmers could use CALL to access this subsystem.

• Enhance the ACCEPT and DISPLAY verbs to provide screen management features such as cursor control.

4.2 Coding Usability

These items allow the programmer a shorthand or more convenient way to code functionality already available.

• Free format source. Eliminate sequence numbers, continuation column 7, area A and area B distinctions, and the identification in columns 73-80.

• Concatenation of alphanumeric literals.

• Allow arithmetic expression anywhere a numeric data item is allowed, such as subscripts, and ADD statements.

• Make FUNCTION an optional word in an Intrinsic Function reference.

• Inline comments. Allow comments on the same line as COBOL statements.

```
MOVE A TO B.     *> The rest of this line is a comment
```

• Nested COPY modules. Allow COPY statement to occur within a COPY module.

• Partial word replacement in COPY and REPLACE.

• Type modelling -- user defined names for structures or pictures.

• VALUE REPEATED. Individual elements in a table may be set to different values or all elements to the same value (as in the Standard).

```
        02    a-table OCCURS 10 TIMES VALUE FROM 1 IS "one", "two"
              VALUE FROM 5 IS "five" "nada" REPEATED TO END.
```

This feature is very useful in initializing tables directly rather than relying on REDEFINES. The REDEFINES method is an obvious "bug breeder".

• GOBACK. Allows programmer to code the same statement in the main and subprogram to end the program. Equivalent to coding:

EXIT PROGRAM
STOP RUN

• Provide a facility to initialize a section of the Data Division. This could increase data integrity.

• SET condition-name to FALSE. SET condition-name TO FALSE and a FALSE value on an 88. This allows a conditional variable to be changed such that the condition-name is true or false without referencing the conditional variable.

```
        01    an-item  PIC X.
        88    cond-1  VALUE "T" WHEN SET TO FALSE IS "F".

        SET cond-1 TO FALSE
```

4.3 <u>Data Structures</u>

These items add new or enhance existing data structures in COBOL.

• BOOLEAN type.

• Operations on sets of records. Provide operations on sets of records in a file or database. A set of records would be determined by a common attribute such as a key value or a range of key values. Standard COBOL verbs would operate on these sets. For example: MOVE FIELD-A TO FIELD-B FOR THE SET.

• Pointers. Pointers are data items which hold a data address.

• Add high level operations for sets, stacks, queues, lists, trees.

• Enhance table handling to allow ALL as a subscript outside of functions. Make subscripts and indices interchangeable. Allow the first element of a table to be other than 1. This would allow vector processing algorithms to be written in a more straightforward way.

• Table sort. An internal sort to sort a table.

```
        02  a-table  OCCURS 20 TIMES ASCENDING KEY IS table-key.
        03  table-key  PIC XXX.

   SORT a-table
   SEARCH ALL a-table ...
```

This feature is especially useful for tables that are going to be used with a SEARCH ALL. It has an additional capability to allow the user to move a table to another that is defined in the same way and have the new one sorted without disturbing the original table (very common in SEARCH ALL or similar uses).

• Commas and currency signs allowed in numeric literals.

4.4 Interaction with Other Standards, Standard's bodies, and Consortia

These items are changes that may be required to interact to other standards. In addition, the COBOL work done by other consortia should be examined. Some work that will impact the COBOL standard are: Common Language Independent Procedure Call Mechanism, Common Language Independent Data Types, Common Language Independent Arithmetic, X/Open COBOL extensions.

4.5 Internationalization

This category addresses problems of writing programs in different countries that have different alphabets,character sets, date formats, ... In addition, the problems of multilingual applications are addressed here.

The largest facility here is a Multi-Octet Character Set (MOCS) handling facility. This would add a new character type to represent characters represented in multiple bytes.

Other items that should be considered for international programming are: multiple currencies in one program, date issues, decimal point or comma, multiple collating sequences and character sets within an application, greater than 18 digits in a numeric data item.

4.6 Interoperability

Interoperability implies connectivity between heterogenous systems.

This category has been broken down into the following subcategories:

4.6.1 Interlanguage call

• Add data types to COBOL from other languages, such as IEEE floating point or C integer type, and varying length strings.

• Add function return to the CALL statement.

• Allow passing: program and paragraph addresses, files.

• Pass parameters by descriptor. Information about the parameters could be passed along with the actual address and value of the parameters would be passed to subprograms.

• Pass parameters by value.

• Allow the programmer to specify in, out, in/out attributes for CALL parameters.

• Allow variable number of parameters on a CALL statement

4.6.2 Subsystem bindings

Specify guidelines for developers of subsytems to use when developing a CALL interface to be used by a COBOL programmer. These guidelines would describe the COBOL types, CALL statement limitations, In addition, add features to COBOL that would facilitate the development of such a facility. Features might include: named constants, hexadecimal literals, bit manipulation, and STOP RUN WITH STATUS.

4.6.3 Cooperative and distributed processing

Investigate the trends in distributed file systems to see if features could be added to COBOL to facilitate programming in these environments.

4.7 I/O Enhancements

These are features that extend COBOL input/output capability.

• DELETE FILE.

• READ PRIOR to read the previous record.

• Dynamic file assignment.

• Timeout facility for I/O statements such as OPEN.

• WRITE FROM a literal.

• Enhancements to facilitate access to input from pipes.

4.8 Major Features

These items are very large topics that could each have been a category by themselves.

• COMMUNICATION module. Should it be made obsolete or modernized?

• User Defined Functions. Provide the ability to write a function in COBOL and to reference the user written function in COBOL programs.

4.9 Misc Language Enhancements

These items are features that fall into no particular category. The usefulness of the features here is not diminished by being grouped into this category.

• CANCEL the storage for an external item.

• Compiler directives. A compiler directive changes the processing of the source program until redirected. Possible compiler directives include: subscript range checking, control of source and copylib listing, and conditional compilation.

• Add new intrinsic functions, such as a function for de-editing.

4.10 Portability Features

These features will improve program portability between operating systems, hardware platforms, and vendors.

• Arithmetic with reproducible results on different architectures. Currently, arithmetic expressions are implementor defined as to the method of evaluation. At times, the same expression may produce different results on different architectures, depending on the precision of the intermediate results that the implementation used.

• Common exception handling facility. The exception processing facility is a method for detecting and reporting exceptions that may occur during the execution of COBOL statements. A common exception processing facility is specified through exception declarations. For example, an arithmetic size error can be detected by specifying a declarative of the form "USE FOR EXCEPTION EC-SIZE". Further information on the cause of an exception can be obtained by execution of additional functions within the exception declaration.

• IEEE floating point or decimal floating point type.

• Portable COBOL subset. Specify a subset of the COBOL standard that would be guaranteed to be portable. This would recognize that all implementations have limits and specify minimums for those limits. For instance, specify the minimum number of parameters in a CALL statement that must be supported, number of data names in ADD. In addition, the implementor defined list would have to be addressed. This might mean that the portable subset would not include any implementor defined items, or some specifications would be added for some items. It might even include programming guideline, such as avoid the COMPUTE statement and/or use the new arithmetic with reproducible results. Another item that needs addressing is the portability of external program items such as program names, and EXTERNAL data items.

• POSIX is a standardized portable operating system interface originally developed by IEEE. A POSIX language binding for COBOL needs to be written. Some features may need to be added such as a method to access operating system environment variables, and a method to access command line variables.

4.11 Programming paradigms

These are features that allow a different style of programming. The facilities being considered in this category are:

• Object Oriented Programming. Add features to the COBOL language to allow an object oriented style of programming. These features would include objects, inheritance, and classes.

• Add recursion to allow programs to program algorithms as they are written in textbooks. A recursive paragraph is a paragraph that directly or indirectly performs itself. A recursive program is a program that directly or indirectly calls itself.

• Add 4th generation language features to COBOL such as the VALIDATE facility. VALIDATE is to input as GENERATE in the REPORT WRITER module is to output. This facility allows the validation of records or data items based on criteria specified in their Data Description entries. The validation can be on the format of the items (numeric must contain numbers, alphabetic letters, etc.), the contents of the items (a range of values, one value, etc.) or the relationship between one item and another. It also has a new type of error reporting mechanism.

4.12 Transaction Processing/asynchronous processing

Transaction processing allows multiple actions upon data. To insure the integrity of the data all these actions must occur or the "transaction" fails and the actions are backed out. Record locking and file sharing facilities and resource recovery and commitment facilities would be included in this category. File sharing is the ability to share files between more than one program. Record locking is the ability to exclude access on a file that is being shared. Possible features / issues: exclusive access, ability to hold locks, deadlock detection, lock a file but allow readers, manual vs. automatic locking.

Asynchronous processing allows non-serial processing.

• CALL don't wait. The calling program could continue executing without waiting for the called program to return.

4.13 Quality Features

These are changes made to improve the documents consistency, readability, and correction of errors and omissions. Some of these items have arisen due to interpretation requests. This category has been divided into 2 subcategories: items not affecting programs, and items that may potentially affect programs.

5 Features most likely to be in the next revision

The committee has taken some straw votes to determine current sentiment for what features should be included in the standard for 1996. This list is in no way a final determination; X3J4 will refine and change this list many times in the coming years. To be included in the standard a 2/3 vote of the committee, not to mention agreement of WG4 the ISO COBOL working group, and numerous public reviews will take place. Items currently on the critical list may not be included in the final standard produced.

The critical features are those without which a standard should or could not be produced. At first pass, the critical features are:

• MOCS -- Multi Octet Character Handling.

• CALL statement improvements to facilitate subsystem bindings and interlanguage callability

• Object Oriented Programming

• Common Exception Handling

• Removal of obsolete items from the standard.

• Conformance to the ISO drafting rules (although this provides little functionality, it will be a major task) .

The important features are:
• Demodularize the standard to improve readability and duplication of specifications. All the I/O modules would become a part of the nucleus module.

• Quality improvements: correction of errors, improvements in the wording of specifications.

• Portable arithmetic including floating point

• FIMS native language support

• Access to operating system environment variable

• Program status return

• Zero length data item fixes

• VALIDATE facility

• User defined functions

• Record locking/file sharing

The desirable features are:
• Archaic category. Features that are in this category would not be enhanced or improved. Implementors would continue to support them as is, and programs that were written using these features would continue to work. In contrast to obsolete items which are deleted in the next full revision, archaic items would continue to be supported but their use would be discouraged.

• BIT/BOOLEAN

• Portable minimum implementation rules.

• Constants.
• Table sort.

• Dynamic file assignment

• Make the word FUNCTION optional

6 Communicating with the X3J4, the COBOL standards committee

There is a great need for input from the COBOL community in the development of the next COBOL standard, especially from the programmer that uses COBOL in their day to day work. There are many ways to get involved: one might join the X3J4 committee, write to the committee recommending features that would be of use to you, or participate in the public reviews of the draft standards. The committee maintains a list of people interested in actively participating in COBOL standards work. To add your name that list write the committee asking that your name be added to that list. You will receive notices of public reviews.

The committee meets six times a year. The meetings are open. Anyone may attend as a guest. All input is considered and responded to. Write the chair at:

Don A. Schricker
Wang Laboratories, Inc.
M. S. 013-790
One Industrial Avenue
Lowell, Massachusetts 01851
Don.Schricker@OFFICE.WANG.COM

or to the author who is also a member of the X3J4 committee.

Interex Paper # 3867

The SLA Cookbook: A Methodology for Managing Computer System Performance

Doug McBride
Hewlett-Packard Company
Performance Technology Center
Roseville, California

Introduction

It seems as though over the last 2 or 3 years, members of the Hewlett-Packard user community have been hearing more about Service Level Agreements (SLAs). The context of the SLA discussion is how SLAs can help in getting a handle on your system's resource consumption and performance. Many of us within HP have been talking about SLAs for some time [1], as have third party suppliers of software and services [2].

The need to better manage computer systems is a reality we need to acknowledge. Significantly larger portions of business "mission critical" applications are implemented on computer systems, and the efficiency with which these systems are managed greatly affect the ability of the enterprise to be aggressive in today's highly competitive markets [3]. Performance and system resource management are a critical piece of this management paradigm. SLAs can play a significant role in helping managers get control of system resource consumption and the resulting system performance.

What is a Service Level Agreement? How can it help you gain better control over your data processing environment? What are the investments you have to make to get a SLA in place and working for you? What are the software tools needed to effectively implement a SLA? Can SLAs be applied to systems with manual components? Can they be used in non-data processing environments? What are the major components of a SLA? Answers to these questions will be addressed in this paper.

It seems as though much talk (and writing) has taken place regarding the use of SLAs, but few sites are using the SLA as a management vehicle. Those that have implemented SLAs in one form or another are reporting success, and have better control over their computing resources. Those that haven't yet implemented SLAs cite lack of understanding of the actual implementation process, although they agree SLAs make a lot of sense and should work if actually implemented.

SLAs can be very simple in scope and not require a lot of management resources to maintain if properly implemented. And, the rewards can be great: increased control over your data processing and network resources; customers who are more satisfied with the end product of your services and who understand what goes into that provision; and better visibility for you, the manager, as you demonstrate established and mature management methods in running your data processing organization within your company.

What I'd like you to be able to do after reading this paper is to feel comfortable with starting down the road to more effective system management using SLAs. SLAs aren't that difficult to put in place -- you just need to do one, and the amount of leverage for those that follow is significant. If you use this paper much like a cookbook -- the right ingredient at the right time -- you should find implementing a SLA to be relatively straightforward.

The material for this paper has been taken from a more robust work of the same title that will be available later in 1992, and will also be presented through HP Education Centers in the form of a one day seminar on managing system performance.

What is a Service Level Agreement?

Service Level Agreements are written agreements between end users (who represent the business aspects of the enterprise), and the company's Data Processing Organization (DPO). The purpose of the SLA is to specify, in mutually acceptable metrics, what the various end user groups can expect from the DPO in terms of system response, quantities of work processed, and system availability [4]. SLAs also specify what the DPO can expect from the various application user groups in terms of system usage and cooperation in maintaining and refining the service levels over time.

Essentially, a SLA is a short document that "puts on the table" what is important for end users to get, in terms of service, from the DPO. There is a separate document created and maintained for each major application user group so that their needs are kept separate from those of other user groups.

The important metrics of success for the user group (in business terms -- more on this later) are identified. The metrics could be OLTP (OnLine Transaction Processing) response times, batch turnaround times for end of day reports, actual hours of system availability, etc. In essence, you're trying to document what this group of workers and managers need from the DPO to best fulfill their contribution to the success of the company. Getting this information established and on paper is one gigantic leap forward from where many data processing shops are today.

From this information the DPO can extrapolate several key items: Can the service levels be met? What's the cost of providing the service levels? What does the DPO need to do to track the metrics? Are all systems automated or are there manual components? What tools are needed? I'll explore these question in more detail later in the paper.

How SLAs Help You Manage

Assuming you establish a SLA for each of your major application user groups, it's a fairly straightforward task to extrapolate the impact of providing those service levels on DPO resources. If you understand how much computational power you need to support the application set, the level of staffing necessary to maintain the DPO to those levels, and to what extent other non-DPO systems are involved in supporting the applications, you've gotten a pretty good handle on what you're trying to manage.

Each application can be characterized by the amount of resources necessary to support it. This is typically referred to as ***workload characterization*** and involves understanding how elements of the application use resources, both automated and manual. When you've achieved this level of understanding, you've established the information base necessary to gain and maintain control of your DPO resources. This is what management is all about, right?

For instance, a well-managed DP shop which employs SLAs would never allow an uncoordinated increase in the number users of a particular application without understanding the effects of increasing that workload's

intensity (the terminology used when making an application work harder by allowing more people to use it). By adding users without understanding the impact of doing so, it's quite possible that the response time service level for that application might be violated. And, it's quite possible that the extra resources consumed by the application to support the additional users could cause other application's service levels to be exceeded. Been there before, anyone?

The techniques used to insure service levels continue to be met make sure workload intensity changes, additional applications added to the system workload, system availability changes, etc. are all handled in a controlled and well understood fashion. Again, this is what proper resource management is all about -- understanding how a change in one place will affect not only that place, but all others. The result is that your end users get their work done in a fashion they specify is acceptable, and you know when it's time to add new resources to handle additional demand.

The result of using SLAs:

- Better management of large capital resources (computer systems and their peripheral complement).

- Higher level of contribution towards the success of the company based upon the more effective use of data processing resources.

- Confidence in the budgetary process by knowing well in advance when expenditures are necessary to meet the demands of the application users.

- Satisfied users of your services.

What Does a SLA Cost?

It doesn't make sense to implement management practices that cost more than they give back, right? So, what are the approximate costs to implement SLAs? The answer to this question is a function of how well you understand your current DP environment and what management practices you currently employ. Let me explain what I mean by that.

What has to happen before you can implement a SLA? Here's a quick list:

- Establish an "inventory" of system resources (CPU, disk space, people, etc.) which service your DP users. This means understanding the available capacity of your "systems" to accomplish work for your users, not necessarily how many computer systems you have sitting in your computer room.

- Out of that "inventory" of system resources, how much is currently being consumed to support the present set of applications?

- What do your application users require to maintain status quo if they are receiving adequate service now? Or, what additional service(s) do they require to get to where they need to be, and how will that affect available system resources?

If you've been using software tools to monitor system performance and resource consumption such as HP GlancePlus or other third party solutions, you probably have a general idea of how your computer systems are behaving. If you've been using a more sophisticated system management tool which includes data logging for later analysis such as HP LaserRX or other third party products, and understand the characteristics of how applications use system resources, the second and third points are also fairly straightforward.

Assuming the manual portions of your systems are also documented, it shouldn't be difficult to extrapolate any increase in service levels against those systems or portions thereof.

If you're not currently monitoring how your computer systems are being used, or don't have a handle on how your manual systems operate, your investment will be, out of necessity, somewhat higher. However, the act of determining these items gives you the information (and hence, the power) to start managing your system in a more effective fashion.

Assuming you are monitoring and tracking your resource consumption data, experience has shown that a knowledgeable technical system manager should be able to assess the first two items in less than a week. This information is leverageable across all application groups on the affected systems so it can be amortized (or optionally goes to zero) after the first exercise.

Each customer interview should take less than 4 hours (after a little prep time -- see below) unless there are serious problems, or you're talking to the wrong person. Assessing what the customer had to say and translating it to resource consumption terms is difficult to quantify due to the variability of metrics. But again, if you have a good understanding of how the customer's application runs on your system, an estimate should be achievable within a day or less.

So, it looks as though a concerted **two week** effort should give you the information necessary to write the SLA, assuming you have tools in place to help gather the necessary data. Adjust your expenditures based upon the level of current management efforts -- but remember, what you end up with is a solid and current picture of ***how your systems are being used, and how a particular application user group perceives its level of service.*** This is information that can be used in a variety of ways, not just to write SLAs.

What about monitoring service levels for compliance? The idea here is to monitor the service level metrics over time for two reasons: one, to show that you are maintaining service levels as agreed to; and two, if a particular service level metric falls outside the agreed upon bounds. We need to look for exceptions to the affected service level metric and report those as soon as they are detected so corrective action can be taken, if possible. Again, more on this later in the paper.

Most of the software tools available from HP and other vendors to collect resource consumption data have the ability to display that information graphically over time. In addition they have the ability to put the data into a form which can be massaged by an external program such as a spreadsheet or statistical package, which can be set up to look for and graph anomalies. The steps to do these are easily documented and take very little time. Additionally, many packages exist which reduce the amount of labor investment to scheduling an event to happen at a specific time, and gathering the output when it'd done on a repeating basis. A weekly look at an application is typically sufficient, unless someone reports problems. This look would contain both the actual tracing of the metric over the week, with exception reports to call out when and where the metric was out of bounds of the service level.

Steps Toward a Successful SLA

Here's a list of things to do, in time sequence, which will result in a successful SLA implementation. Because of the variance of individual situations, you may have to modify some of the steps to meet your specific circumstance, but the general flow is still the same.

1. Establish your resource inventory.

You have to know what you've got in terms of overall ability to provide service before you can really get started. The types of things you're interested in are both computer and non-computer specific. For example, you need to know the aggregate amount of CPU power available for applications. In simple cases, this may be a single box, or in other cases you may be dealing with multiple boxes, multiprocessors, or client server situations. Available disk space, printing capability, terminal/workstation connectivity, off-line storage for backups; all are important and need to be quantified.

You also need to understand how the non-computer parts of your applications function. It is a rare system indeed that doesn't have a manual component somewhere in the process. The relationship between these resources and automated facilities needs to be well understood. Too many times motivated DPO managers have optimized computer resources only to find the application bottleneck was in some manual portion of the system!

Exit criteria for step 1 should be a detailed list of system resources and configurations which can be referenced during the remainder of the study.

2. Understand how resources are currently being used.

Although somewhat related to item 1, understanding in detail how your resources are being used insures you can account for where your capacity inventory is being consumed. This should be understood from both a global (system-wide) and per-application basis. The global perspective gives you a feeling for what might be left to use to improve existing service levels, or to establish new applications. The per-application audit establishes a firm baseline from which you can make additional extrapolations as to how an individual application consumes resources which provide the resulting service levels. Application information is also critical to have when entering the next phase of SLA development as you'll soon see.

In order to understand how your computer resources are being used, you need some kind of software tool that will give you information in a way which is easily interpreted. These tools collect and display information gathered from the bowels of the computer's operating system, from instrumentation that was placed there by the OS designers to help understand how the system is being used

Many tools present this information as a lump of tabular data. Although tabular data is necessary to discover discrete events, large listings of numbers by themselves are virtually meaningless in this environment because you're trying to determine service level metrics at a much higher level. Looking at reams of numbers has a tendency to numb you to what you're actually looking for, and is the wrong approach to this type of problem.

Graphical representation of this data is much more effective and gives you the "big picture" needed to manage at this level. Although tools which display immediate results of real-time data are somewhat helpful here, the real need is to be able to see system resource consumption data, both global and application displayed over time. This allows you to see both instantaneous and short term events to observe "burstiness", and also to get a feel for trends of resource consumption over a longer period of time.

Tools such as HP's host-based SCOPE collector fall into this category. It allows you to collect system global, user-defined application, and individual process data for analysis using your own mechanism (spreadsheet, statistical package, etc.). HP LaserRX is a PC Windows graphical user interface which shows the SCOPE data in a variety of scenarios if you don't want to spend the time developing your own analysis mechanisms. Other vendors produce similar tools which can also be used in a similar fashion.

The bottom line is you can't, in any effective fashion, get a handle on this information without the assistance of tools which collect and optionally display this data. Historically, in other computer vendor environments, the management of system resources started to occur only when tools of this nature were made available. Without these tools, the people resources spent on management is too high, and the return on investment too unpredictable to make the activity worth doing. The same is true in the HP environment.

Exit criteria for step 3 should be a write-up of your findings, with supporting graphs and tabular data. The global perspective should be addressed separately from the application perspective. Each application should have a brief writeup to cover the computer system and manual resources used.

3. Establish current service levels for a given application.

Prior to engaging the application user group for a discussion of their current service levels, you need to establish, in general terms, what the current service levels actually are. Often times this is quite easy, especially if you already have a good working knowledge of the application and the way in which it is used.

It behooves the DPO staff working on the SLA to know as much about the application and it's associated system requirements as possible before going to work with the end users. This is not to imply (by any means!) that the DPO is keeping the SLA process a secret. Quite to the contrary, it's important to get conceptual buy off before going to any great lengths on the project. However, if you don't have a handle on how their application is behaving, you'll not have all the information you need to do a good job on the next step.

If information on the application user group's current service levels is unknown, steps need to be taken to get the data. Customer interviews, meetings with application support staff, review of working flow diagrams, etc., should establish the metrics. Your measurements from step 2. should help give you the actual values.

Exit criteria from step 3 should be a concise description of each application and its current service levels. The metrics being described should have been ascertained by working with the user group, or from expert knowledge based upon working with the application in the past. Focusing on the wrong metric(s) for service level management is a waste of time.

4. Determine customer needs (and wants).

Up until now, we've just been gathering the resource consumption information necessary to start working on the SLA. Now that we're armed (and dangerous!) with the data we need to work with the customer on getting the SLA set up, we can sit down with representatives of the application user group, and see how things are going. The customer has obviously (as mentioned in the previous step) been primed for what is to take place in the meeting, so what you're trying to determine is how effective the DPO has been in satisfying the customer's needs and wants.

Just a side note here. It's the DPO's job to discuss the application with the user(s) in their (business) terms, **NOT DP** terms, and to translate what the user says into terms the DPO can then use to measure against metrics. Think of all the times you felt at a disadvantage when someone was speaking to you in jargon about something that was very important to you (financial plans, car, house, etc.). You'll really establish yourself as a professional if you go to the lengths necessary to understand the user's needs in **their** terms, and you'll get much better answers than if you try to elicit their answers in your terms instead.

Your interview team needs to ascertain if the current service levels are satisfactory. If not, what are the acceptable service levels? Why does the user perceive current service levels are not adequate? (In most cases, this is already known based upon past problems, and a degree in rocket science isn't needed to figure out the current situation.) What, in their opinion, can the DPO do to correct the deficiency?

There is, of course, the situation in which there are insufficient resources to achieve the desire service levels. You should know this from steps 1 and 2, with the details refined in step 3. This should not be a point of detailed discussion during the customer interview, however. You may find it difficult to even quantify an absolute answer without further study and extrapolation. More on this in step 5.

As I mentioned before, there is the issue of **needs** versus **wants**. **Needs** are what is required to get the job done per minimum specs. **Wants** on the other hand, are usually a superset of needs and although important to user satisfaction, should come with a higher price tag or set of tradeoffs in order to reflect their non-critical aspects to getting the job done. On the other hand, politically speaking, addressing wants can be just as important as addressing needs.

The exit criteria for step 4 should be consensus from both DPO and the application user group that the proper metrics have been identified and will be used, in one form or another, to track service level compliance.

5. Determine the costs of providing the service levels.

Here's where the SLA starts to give some payback.

Since you've identified the resources consumed to support current service levels in step 2, you need to attribute the cost of these resources back to the user (although not necessarily for the purpose of cost recovery). Most managers who use Chargeback Accounting systems should already have a good handle on this information, or be able to break it out rather easily.

If you are running your DPO as an expense center for the enterprise, you may have a harder time in quantifying these costs due to the lack of an established unit accounting and rate structure. You need this data primarily to help determine the costs of providing higher levels of service. What you may find out in the process however, is that you are under- or over-charging for the service now being provided. That's good data any way you look at it.

Ideally, where applicable, you should strive to get the costs of the service levels down to a **business transaction** level. This allows the quantification of the costs the end user can understand, rather than discussing costs in computer work units, or whatever. In all cases, costs should be broken out by line item -- that is, if there are several factors which affect the cost of the service, they should be called out individually as well as in the aggregate.

You or your staff should be able to take these customer-oriented service levels and translate them into **Service Level Objectives (SLO)**. This is the act of turning a logical perspective of the application into its actual physical components which can be monitored by the DPO. For example there may be disk space or a volume mounting component built in to the service level criteria. System availability plays a big role in service level management, as does network connectivity. In many cases, due to lack of business transaction data, extrapolations to more primitive measurements may need to take place and the relationships between the higher level business transactions and the lower level computer and manual system activities defined.

It is understanding the SLA/SLO relationship that eases the costing and service level monitoring chore.

Exit criteria for step 5 are service level costs associated with each service level metric for the particular application.

6. Determine if desired service levels can be met or maintained.

As mentioned in step 4, it may not be possible to meet or maintain the desired service levels with current system resources and current system demands by all applications. You can't agree to maintain service levels if the resources aren't there to meet them, so it's necessary to make this call before writing the SLA.

What do I mean by "maintain" current service levels? You're meeting them now, right? This may be true, but very few systems are static in nature. Even if the application you're studying is not changing in any significant way, other applications which share system and enterprise resources could be. **The contention for these resources is what determines the service levels in the first place, so you need to understand how other applications are behaving to ensure service levels for the application of interest can be maintained at status quo.** And, what's the cost (from step 5) of maintaining the service level?

If desired service levels are not being met, you have to extrapolate what additional resources will be needed to achieve that service level. There are many ways to do this from using mathematical models to pure WAGs. Most managers can find a happy medium and an acceptable answer by simple back-of-the-envelope computations. This is made all the more easier by having done a thorough job in steps 2 and 5.

Once the resource requirements are established, costs can be identified and documented as to what it will take to achieve the desired service levels. It is up to the manager as to whether he or she wants to use the SLA development process as a forum for acquiring the additional resources, however I would advise against it since that's not the intent of the SLA process -- just a valuable by-product. Since the application users are receiving a certain level of service at the current time, for the purpose of the SLA, I'd suggest using those values as what you'd like to maintain, then use the opportunity of providing the additional resources to meet the needed service levels as a time to re negotiate the SLA.

Exit criteria for step 6 is assurance that current or desired service levels can be met, for how long, and the costs associated of bringing resources on-line to meet perceived deficiencies in current service levels.

7. Write the SLA.

Never thought we'd get here, did you? Granted there are several steps to get to this point, but now we've gotten all the information we need to write the SLA. Although there are many components to the SLA [6], not all are required. Here's what I would consider to be the minimum content/format of a SLA:

- **Background.** There should be enough information imparted within this section to inform a reader who is less than intimate with the application in question to understand current service levels and why they are an important business indicator.

- **Parties to the Agreement.** Who are the players in this SLA? Who is the responsible party with the DPO? Who is the counterpart in the application user group?

- **Volume of Service to be Provided.** It's unreasonable to expect the DPO will provide an infinite supply of the service needed by the application user group. This section **quantifies** the volume of the service to be provided. The application user group should be able to specify the average and peak rates, and the time of day the demand is expected to occur. The user may also be provided with the incentive to receive better service, or a reduced cost for service, by avoiding peak resource usage periods. This metric is used to help the DPO project application user needs, as well as limiting the amount of work the application user can try and do during a given time interval.

- **Timeliness of Service.** This is usually the metric most people focus on for SLAs, although as you can see now, it's not the only one. It's essentially the **qualitative** measure of most applications because it dictates how quickly the user can expect to get their work accomplished. For OLTP applications it could be "90% of transactions processed within 2 seconds".

For more batch oriented applications, it could be "reports to be delivered no later than 9:00am if input is available by 11:00pm the previous evening".

- **Availability of Service.** When will the service be available to the user? The DPO must be able to account for both planned and unplanned system unavailability. The users must be able to specify when they expect the system to be available in order to achieve their specified levels of work.

- **Limitations of Service.** The DPO cannot support an environment of unlimited resources. There are certain limits to the service a user can expect due to peak period demands, resource contention by other applications, and general overall application workload intensities. These limitations need to spelled out explicitly and agreed to by all parties so as to not cause finger pointing and unhappy customers down the road.

- **Compensation for Service.** If SLAs are expected to be taken seriously, and if any teeth are to be put into the agreement for both parties, some form of compensation needs to be established. The ideal situation is a real chargeback system by which end users are charged for the resources consumed to provide the service they expect. Higher levels of service typically have a greater cost associated with them, and this gives the end user the opportunity to apply his or her own management methods to try and optimize DP costs to get their work done. If a real chargeback system cannot be implemented for practical or political reasons, the costs should still be identified and reported back to the user and upper level management to show where the DPO resources are going. The timeliness and format of this information should be described here.

- **Measurement of Service.** This section describes the monitoring process by which actual service levels will be compared against the agreed upon service levels. This will be covered in more detail shortly, but the intent is to define how the service levels will be monitored, and the frequency with which normal monitoring will take place. A brief description of the data collection and extrapolation process is appropriate, as well as how user perceived problems are to be reported to the DPO.

- **Renegotiation of Service.** The SLA should be a living document. As time goes on, application usage may change, additional applications may come on line which affect levels of service, etc. Strong communication between the DPO and the application user groups is imperative for both parties to be successful. This item makes sure that after a certain period of time, if it hasn't already been done, the SLA is examined for its effectiveness and redone if there are changes which need to be reflected in the document.

After the SLA is written, it should be reviewed by all applicable parties and issues resolved. After consensus is achieved, an implementation meeting should be held and actual details discussed along with responsible parties and necessary actions.

8. Monitor the Service Levels.

Once the SLA is in effect, a monitoring process should be put in place to ensure compliance by both parties to the agreement, and to help spot trends which may indicate consistent out of tolerance service levels. Essentially this dictates two types of reports; normal and exception.

Normal reporting mechanisms should be able to show the value of the service level metric(s) over time, and be available for discussion in case of disputes. The source of the service level metric data should have already been established in the preceding steps, so it's just a matter now of developing a non labor-intensive mechanism to collect and plot the data. As I mentioned before, many of the tools currently available from HP and other vendors support some feature for getting data into a display or statistical package, or have a display package which can be purchased specifically for that purpose. Monitoring of non-automated service levels and their associated objectives can be implemented by addendum to existing processes used by the manual systems to collect and record that data for later use. In many cases, this data is easily integrated into display packages that are being used to report on the automated metrics.

Exception reporting should be implemented for two reasons:

1. It provides a proactive mechanism for the DPO to see when service levels are out of tolerance. If the cause of the exception (response times greater than the allowable percentage, reports delivered late, etc.) is under the

control of the DPO, corrective action can and should be taken to get the service levels back in line with the SLA. If the user group is contributing to the exception condition, the DPO can notify responsible parties early in the cycle and work with the users to fix the anomaly.

2. It gives an indication early on (if the exceptions continue) that a trend is developing and corrective action needs to take place. This corrective action may be in changing the way the application users consume resources, a load balancing exercise with other applications, or the planning necessary to acquire addition resources to meet the increased demand.

Again, most tools available to monitor computer resources and help with this exception reporting. If manual data can be fed into a spreadsheet package, exception reports can easily be generated along with graphical representations of the data to help support the issue.

Summary

In this paper, we've explored in detail how to implement a Service Level Agreement within your data processing organization. As you can see by the steps necessary to implement a SLA, and the data sources used to feed the SLA process, increased management control of data processing and associated manual resources are an instant fallout.

SLAs in and of themselves don't make end users happier with the DPO, nor do they improve system performance or availability. What they do is make DPO management (and I'm sure you can see these techniques are not just limited to data processing) asses how their systems are being used, quantify the costs associated with using those systems, and plan for future growth to meet current and future needs. That's what management is all about, isn't it? Taking responsibility and implementing planning and control.

As a sage philosopher once said, his or her identity lost for all history:

"If you are aiming at nothing, don't be surprised if that's all you hit".

References

[1] McBride, Doug, "*Service Level Agreements: A Path to Effective System Management*", HP Professional (August 1990)

[2] Duncombe, Brian, "*Service Level Agreements - Only As Good As The Data*", Conference Proceedings of the HP International Users Group, (August 1991)

[3] McBride, Doug, "*Performance is a Dirty Word Around Here...*", Conference Proceedings of the HP International Users Group, (August 1991)

[4] Salminen, Veli-Matti J., "*Leveraging Service Level Agreements*", Conference Proceedings of the Computer Measurement Group (December 1989)

[5] MacKinnon, Douglas, "*Service Goals: Who Needs Them?*", Conference Proceedings of the Computer Measurement Group (December 1986)

[6] Miller, George W. (Bill), "*Service Level Agreements: Good Fences Make Good Neighbors*", Conference Proceedings of the Computer Measurement Group (December 1987)

PAPER #3868

Enterprise Performance Management - By Exception Only

Wayne Morris
Hewlett Packard Company
Performance Technology Center
8050 Foothills Boulevard
Roseville, CA 95678

Abstract

Managing the performance of a large distributed heterogeneous environment (also called enterprise performance management) is not an easy or simple task. It requires new capabilities, techniques and management concepts. The performance manager in this environment cannot afford to wait for user complaints or to actively monitor all nodes in the network.

This paper examines a management-by-exception approach to on-line performance management of open system environments. Underlying technology and management techniques are outlined and illustrated through the use of examples.

Introduction

The objective of performance management within an organization is to accomplish the following:

- ensure user productivity is not negatively impacted by the service levels (response times, job turnaround times) provided by the computer facilities;
- optimize the return on investment in the computer facilities;
- plan for support of future business goals and strategies by the computer facilities.

Computer facilities refers to the computer network, systems and applications that together automate one or more aspects of the organization's business. It is important to recognize that enterprise performance management encompasses all system platforms as well as the network devices and connections.

Performance management is the science of optimizing the way in which the business uses the computer facilities in order to meet the objectives outlined above. This involves characterizing the inherent performance capabilities of these facilities, optimizing the usage and availability of the facilities, and predicting future usage and capacity requirements.

Traditionally, performance management has been reactive in nature. Computer facility users experiences slow or erratic system response, or perhaps their batch jobs haven't completed within an acceptable time, and they notify the help desk or system manager of the problem. Waiting for a problem to be identified by the user community implies that user productivity is already impacted before we begin to diagnose and resolve the problem, and consequently the profitability of the business can also be negatively affected. Because of this, the problem resolution will probably be undertaken in high-priority crisis mode, increasing the pressure on the system manager or performance analyst.

This leaves little time for the more proactive aspects of performance management such as examining workload patterns, scheduling jobs to take advantage of periods of low usage, and balancing utilization levels within classes of resources. Operating in a reactive mode also means that capacity planning is rarely undertaken, and hence as the business changes, it is probable that the computing facilities will not be able to support new requirements such as additional applications, users and workloads.

When compared to the stated objectives for performance management, this traditional approach is not really successful. Also there is very little time to plan for future requirements because of the reactive nature of performance management. Performance management can be made more efficient through the application of management-by-exception concepts and techniques. The remainder of this paper examines these techniques and illustrates their use with the help of examples.

What is Management-by-Exception?

Instead of waiting for users to be affected by performance problems, or actively monitoring all of the resources and applications within the computing facilities, we can focus on areas requiring attention by looking for deviations from a normal or desired state. This technique, referred to as management-by-exception, is not a new concept and has been applied to many management areas. When implemented for performance management, it generally involves some form of alarm notification based on pre-determined thresholds being exceeded. This concept is particularly important when managing large numbers of systems within a distributed environment where it isn't viable to examine performance on a system by system basis.

Depending on the alarm technology employed and how the thresholds are set, this technique can warn of potential problems before users experience difficulty. Instead of waiting for user complaints, we can identify, characterize and correct problems before user productivity is affected, thereby achieving our first objective for performance management. Because we are focusing only on those areas requiring attention, this technique can also help lower our management overhead. This allows the performance manager to become more proactive by planning for future capacity requirements.

When does an exception exist?

The first step in determining if an exception condition exists is to define the normal or desired state. To do this, we need to examine the goals and objectives of the business, and in particular, to take the user's perspective of performance. Users will generally know when the computing facilities impact their productivity, but may have some difficulty in clearly articulating their requirements. The process of interviewing users to determine and document the required service levels is part of the process of establishing Service Level Agreements (SLAs). More information on service level agreements can be found in [McBRIDE] and [MILLER].

An SLA documents the user requirements and services to be provided in terms of application and system availability, response times, job throughputs, quality measures, anticipated workload volumes and user-projected growth in applications and workloads. This provides for a clear understanding of the services to be provided by the computer facility, and allows the facility managers to determine a set of measures and goals that defines the desired or normal state. These service level objectives are typically defined as:

- response times will be less than or equal to *A* seconds *B*% of the time
- batch jobs will complete within *C* minutes from submittal *D*% of the time
- the system will be available *E*% of the time

Each of these measures will normally be defined on a per-application basis and may also specify the measured shift. For example, the response time objective might be stated for the prime shift rather than across the entire day. An application is a grouping of programs or processes which automate one aspect of the business such as payroll or order processing. The service levels required for a financial application may be much higher than those required for an inventory control application, and so the objectives reflect the relative importance of each application to the organization.

Another consideration may be optimizing the utilization of resources within the computing facilities. Under certain conditions, this goal can be contradictory to maintaining user productivity, and hence this may become a secondary objective. We may define the secondary aspect of our desired state as:

- resource utilization will not fall below *A*% more than *B*% of the time

This measure would be required for each class of resource and would be measured globally rather than on an individual application basis.

Now that we have defined our desired state, the next step is to continuously monitor the actual performance of the computing facilities to determine when there are significant deviations from normal, otherwise known as exception conditions.

How do we recognize an exception condition?

As the performance measures are collected, we need to be continuously comparing against our stated service level objectives and resource utilization objectives to determine if an exception condition exists. The simplest form of determining when an exception condition exists would be to compare our measured performance in terms of response times and job throughputs against our service level objectives, and when the objectives were exceeded to trigger an alarm. This sounds too easy to be true, and the situation is generally more complex than this. This is particularly true of UNIX-based systems, where the definition of response time is very inconsistent.

Users measure service levels based on what they experience on their desktop. In most cases the performance monitors measure response time as viewed by the system, and depending on the connection between the user's desktop and the system, this may not accurately represent response times perceived by users.

Response time measurement becomes particularly complex in distributed client-server applications, when the user measures response time for a complete transaction representing a unit of work which may require several interactions with their desktop and, potentially, some interaction between their desktop and one or more servers in the network. New techniques are being developed to accurately measure end-user response time in distributed environments. Until these techniques become generally available, the solution is to instrument the application code, or to correlate system measured response times and / or resource utilizations with user-perceived service levels.

Job turnaround also presents difficulties for performance monitors. Again the most accurate measure here is to instrument the critical batch applications. If this is not possible, a global measure of job turnaround is usually available on proprietary systems, or you may be able to correlate job turnaround with resource availability for batch jobs.

Instrumenting the code is possible if the application has been developed in-house, and normally involves time stamping when each critical transaction or batch job commences and completes. This can then be summarized and periodically written into the performance logfile.

It is important, when identifying exception conditions, to ensure that false alarms aren't generated due to transient "spikes" in performance. Therefore we need to include a time factor in our exception finding logic. This time factor can be calculated from the service level objective by looking at an average over a period of time such as five or fifteen minutes rather than examining a shorter time interval. For example if our service level objective says that response time should be less than 2 seconds 95% of the time, we shouldn't create an alarm if response time is 2.5 seconds during a one minute interval. However if response time remains at 2.5 seconds for ten minutes, then this would normally constitute an alarm condition.

To expedite problem resolution we should also assign a priority and severity with each exception condition. The priority reflects the importance of the application, transaction, user or system to the business, while the severity indicates the extent of the problem. These categories will help determine the order and urgency for addressing and correcting exception conditions. Applications are generally defined in a parameter file as a grouping of processes, or user sessions. The priority for each application would also be specified in this file. Severity would be dynamically assigned according to the extent of the deviation from our service level objectives or by the significance of the performance bottleneck as determined by resource utilization levels or a combination of bottleneck indicators.

For example, a memory bottleneck could be determined by looking at the following indicators:

- CPU percent used to manage main memory
- Physical disk rate for swapping (Memory management)
- Average number of processes waiting on main memory resources

Multiple thresholds could be assigned for each of these metrics to indicate bottleneck severity. As each threshold level for a single indicator is exceeded and as the thresholds for multiple indicators are exceeded, then the severity of the bottleneck will be increased.

The management-by-exception concept has been successfully employed by a number of integrated network management systems (INMS) such as HP's OpenView, and IBM's NetView. Performance management applications for both single systems and distributed environments are now becoming available which also utilize management by exception. In some cases these performance applications are integrated with the network management facilities to provide an integrated network, system and performance management for the complete environment. This integration can be very beneficial, particularly when dealing with large numbers of systems.

When selecting or building an enterprise performance management application, it is important to ensure the application is both extensible and open. Most organizations have multiple hardware vendors, and multiple platforms from each vendor. Because of this the performance management application has to be able to monitor and manage these heterogeneous systems. It is unlikely that a single vendor will be able to supply performance monitors for every platform within your enterprise. Therefore the enterprise performance management application needs to provide a framework with the capability to integrate multiple vendors' performance monitors and system-specific performance management applications. In this way, there is a single view for all the exception conditions within your environment. When you have isolated a problem and wish to conduct detailed analysis, you can use the most appropriate system-specific performance monitor from within the same user interface. Another requirement is openness of data. You may wish to conduct custom performance or capacity analysis, and hence open access to the data collected by the performance monitors can be important.

The most effective way of conveying the status of the entire environment is to represent all monitored nodes and network devices as icons on a logical map. The integrated network management systems can provide this mapping capability as well as automatically discovering all nodes in the network. Exception conditions can then be shown by highlighting the node.

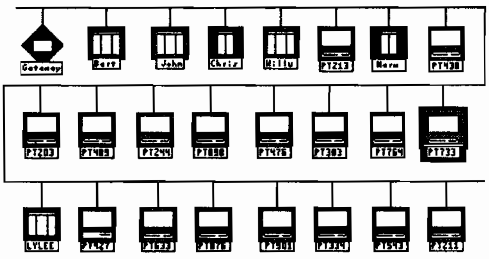


Figure 1.

Figure 1 gives an example of such a representation for a bus segment where the node PT733 has an outstanding alarm. By integrating a performance application with an INMS, network, system and performance alarms and events could all be shown on a single view of the network. Depending on the organization structure and the problem handling process, several logical views may be desirable. These views would be tailored to the particular manager, for example a system manager may not be interested in the network's physical topology. Most INMS products can support logical views of the environment. The trend toward integrated network and system management will increase as the OSF Distributed Management Environment and compliant applications become available.

What do we do when we recognize an exception condition?

Once we have identified an exception condition, we need to determine the cause and take corrective action. The first step is to isolate the problem to a particular system and, if possible, application. We then need to characterize the nature of the problem and determine which processes, users and resources (either system or network) are affected.

The complexity of this isolation phase will be determined by the computing facility's configuration. A problem within a large, complex, distributed heterogeneous environment may take considerable effort to characterize and isolate, particularly if client-server applications are involved.

An example of a logical approach to isolating an exception condition is:

- Are there one or multiple exception conditions within the environment?
- Are multiple systems involved?
- Is it one application, inter-related applications, or completely independent applications?
- If multiple exceptions are related, is there one system acting as a server which is the primary cause of these alarms?
- What are the priorities and severities of each of the primary alarm conditions?

The intent of the above is to isolate each of the primary alarm conditions and to determine the order in which each exception condition will be addressed. Each primary exception condition can then be addressed by examining the alarm condition details including the service levels for the application (if the alarm was triggered by an application), and performance measures for the system resources. The goal is to identify which system resources are the current performance bottleneck, which processes are most heavily using these critical resources, and which processes (and hence applications) are affected by the performance problem. This should then allow us to explain why the exception condition occurred, and to make recommendations to correct the condition including the processes and system resources requiring corrective action.

General performance "rules of thumb" are available for most systems, from either the hardware or application software vendor, or third party consultants. These rules can assist in determining which resources comprise the bottleneck, however it is important to realize that to be most effective, the rules should be tailored to your own particular environment. The ability to tailor the rules will come with experience built by monitoring the performance patterns of your environment under your applications and workloads over a period of time. In a similar fashion, performance rules can also be ascertained for your network devices and topology.

These performance rules will allow the critical resources or system bottleneck to be identified for the particular exception condition under scrutiny. Individual processes can be examined to determine which processes are the heaviest users of the critical resources. Then, based on the relative importance of the processes involved (both the heavy resource users, and those experiencing performance degradation), corrective action can be specified. The corrective action could involve altering process priorities, or perhaps even cancelling and rescheduling a particular process. Note that the immediate concern is to relieve the current condition, but there may also be other longer-term actions required, such as increasing system resource capacity, balancing workloads, or reducing workload demands by altering job schedules or tuning the application itself.

The capacity planning process is a separate topic to the on-line performance management concepts and techniques discussed here, however effective use of management-by-exception can point to those system and network resources which potentially have capacity problems. This will assist the capacity planner to be more focused, particularly in large distributed environments.

So far, we have defined a process that starts with defining our desired service levels and identifying violations of these. However, because we identify the problem only after service levels aren't met, user productivity will already be affected. One of our objectives is to ensure that user productivity is not negatively affected, and so we need to improve on the techniques outlined above.

How can we be more proactive?

We could set our service level thresholds lower, in order to provide a "buffer" between our alarm notification and any impact on user productivity. Depending upon the performance characteristics of the system or network, this could mean providing excess capacity which would never be utilized, and hence we wouldn't meet our second objective of optimizing the return on investment in computing facilities.

There are a number of alternative techniques, based on management-by-exception, that can more effectively identify potential problem situations. Rather than simply examining the current service levels and resource utilizations against fixed thresholds, we could build and continuously maintain a performance profile by hour of day and day of week. We would then monitor for statistically significant deviations from the profile, which provides a more accurate representation of exception conditions.

Monitoring service level trends over time would allow us to characterize and quantify performance degradation trends and therefore project when service levels will no longer be acceptable. This could be accomplished by retaining the performance measures in logfiles, and then periodically using time series analysis to identify and project trends. Exception conditions would be based on exceeding our service level objectives within a specified future time period. Time series analysis may not always be able to predict future trends, depending on the nature of the workloads and service level patterns. In this case more sophisticated analytic modelling technology could be utilized.

In order to accurately identify potential problems, we will probably need to use a combination of the above methods, as well as examine a number of system performance and resource utilization metrics. For example, if we determined that we were approaching our service level objectives and that the critical resource was the system CPU, we would want to examine which processes were heavily using the CPU, which were CPU starved, how many processes were queued up waiting on the CPU, the relative priorities of these processes, and their relative importance to the business. The fact that we are approaching our service level objectives for interactive sessions and that CPU utilization is high, might not really present a problem if the primary CPU user is a batch process whose priority is lower than our important interactive applications.

Over-utilized system and network resources will be identified when utilization thresholds are exceeded. These resources could then be examined in more detail to determine if workload balancing would relieve the situation or if additional capacity is required. To do this we will probably need access to longer histories in order to completely characterize the resource utilization patterns. This information is available from resource management applications such as HP LaserRX. Similarly, we may also want to identify under-utilized resources within our environment which could perhaps be redeployed to support more critical applications.

Where do the performance measures come from?

All of the above assumes that performance metrics are available and can be continuously monitored. Monitoring the performance of the computing facilities requires access to performance metrics captured from operating systems, intelligent network devices, and potentially from applications. In the case of systems, this requires the operating system to be instrumented to provide performance information about processes, applications (groupings of processes), and global resource utilizations. A number of proprietary operating systems have been instrumented and performance monitors are available which collect and log these performance metrics. In the UNIX* arena, interfaces such as /dev/kmem are generally supported which also provide performance metrics. However these metrics may not be totally validated and may not be as comprehensive as those provided on proprietary systems.

The performance collection should provide the required detail of information, and the collection and logging process itself shouldn't impose too high an overhead on the system. It is also important that the performance metrics allow service levels to be evaluated from a user perspective. Users are typically concerned with an application or particular transaction within an application. Therefore, the collection mechanism should support the grouping of process information into defined applications that correspond to user terminology such as the payroll application or order processing application.

Similarly a mechanism by which a user transaction can be defined, measured and reported may also be important in determining if user service levels are met. This mechanism could either be provided by allowing application transactions to be marked by an operating system intrinsic, which can then be identified and logged as part of the normal collection process, or by allowing entries to be written into the performance logfile directly by an application. As indicated previously, this ability is very important for UNIX-based systems, and for distributed client-server applications.

Another important consideration is where the alarm logic resides. A number of network management systems hold the performance metrics for each intelligent network device remotely, but then periodically transfer these metrics to a central analysis system where they are examined to determine if exception conditions exist. This works well for network devices with a small number of metrics, but is not really viable for large numbers of systems. The number of processes and metrics per-process we need to examine would mean that the network and possibly the central system would be overwhelmed by the amount of information to transmit and examine. Therefore, the intelligence to identify exception conditions should reside on each monitored node.

When an exception condition exists, the monitored node would notify the central management system of the condition and also send supporting information. This state is referred to as "loosely monitored". The central manager may then wish to bring a node with outstanding alarms into "closely monitored" status. At this time a connection would be established between the management system and the remote monitored system, and all of the performance metric information for that system would be available to the central manager.

The number of metrics and their level of accuracy and validation will vary from platform to platform, so managing the performance of multiple heterogeneous systems is more difficult than managing a homogeneous environment [MORRIS]. To address the needs of a standard measurement base, several vendors have formed a Performance Management Working Group to develop the requirements for a performance management subsystem for the UNIX operating system. This group also includes two major developers of open systems based platforms: Open Software Foundation (OSF) and UNIX International (UI). The goal of this group is to help define standards for performance measurement interfaces and metrics.

Figures 2 and *3* illustrate how metrics can vary from platform to platform. They provide sample lists of metrics for both the HP 3000 running the proprietary MPE operating system *(Figure 2)* as well as for the HP 9000 running HP-UX *(Figure 3)*. These global metrics would typically be of interest when attempting to isolate performance problems.

Additional metrics are also available for each defined application. As previously mentioned, applications are generally defined in a parameter file by specifying program files, users, and process priorities or a combination of all three. More detailed process oriented metrics would probably be required to resolve a problem once it had been isolated to a specific node. These metrics are captured and are available through diagnostic performance management products such as HP GlancePlus.

Sample Global MPE Performance Metrics

TOTAL CPU	Overall CPU busy percent
SYSTEM CPU	CPU percent used by system processes
SESSION CPU	CPU percent used by interactive sessions
BATCH CPU	CPU percent used by batch jobs
MEM MGR CPU	CPU percent used to manage main memory
CACHING CPU	CPU percent used for disc caching (MPE/V only)
DISPATCH CPU	CPU used to dispatch processes (MPE/iX only)
OTHER CPU	CPU percent used for categories other those listed
CPU PAUSED	CPU percent idle, but waiting for disc IO to complete
IDLE CPU	CPU percent idle, not waiting for disc IO to complete
PHYS DISK	Overall physical disk IO rate (IOs/second)
SYSTEM DISK	Physical disk rate for system processes
SESSION DISK	Physical disk rate for interactive sessions
BATCH DISK	Physical disk rate for batch jobs
MEM MGR DISK	Physical disk rate for swapping (memory management)
LOGICAL DISK	Logical disk rate (satisfied from memory or disk)
DISK UTIL	Average utilization for all disk drives on the system
ANY LOGL IO	Peak logical IO rate for any drive on the system
ANY PHYS IO	Peak physical disc IO rate for any drive on the system
ANY MEM IO	Peak memory management IO rate for any drive
ANY UTIL	Peak utilization percent for any drive on the system
CPU QUEUE	Average number of processes using or waiting on the CPU
DISK QUEUE	Average number of processes waiting on disk IO to complete
MEMORY QUEUE	Average number of processes waiting on main memory to complete
IMPEDE QUEUE	Average number of processes waiting for software locks
NUM JOBS	Average number of batch jobs logged on
NUM SESSIONS	Average number of interactive sessions logged on
ACTIVE JOBS	Average number of batch jobs that were consuming CPU
ACTIVE SESS	Average number of interactive sessions that were consuming CPU
TRANSACTIONS	Terminal transaction count since last update
FIRST RESP	Average first response time for terminal transactions
RESPONSE	Average time to prompt for terminal transactions

Figure 2.

Sample Global HP-UX Performance Metrics

TOTAL CPU	Overall CPU busy percent
SYSTEM CPU	Percent of time executing in system code
USER CPU	Percent of time executing in user code
SYSTEM CALLS	System call rate average per second
PHYS DISK	Overall physical disk IO rate (IOs/second)
VM DISK	Physical IO rate for virtual memory activity (paging and swapping)
FILE DISK	Physical IO rate for file system
DISK BYTES	Average physical bytes transferred to and from disks
DISK UTIL	Peak utilization for all disk drives on the system
CACHE HITS	Percentage of file system buffer cache hits
MEMORY UTIL	Percentage average utilization of memory
SYSTEM MEMOR	Percentage average utilization of memory allocated to the system (kernel)
USER MEMOR	Percentage average utilization of memory allocated to user code and data
SWAP UTIL	Percentage average utilizaiton over all swap areas
PACKETS	Average rate per secoind of inbound and outbound LAN packets
ERRORS & COL	Average rate per second of LAN packets collisions and errors
NFS CALLS	Average rate per second of inbound and outbound NFS calls
DUX CALLS	Average rate per second of inbound and outbound DUX calls
CPU QUEUE	Average number of processes using or waiting on the CPU
DISK QUEUE	Average number of processes waiting on disk IO to complete
MEM QUEUE	Average number of processes waiting on memory resources
NETWORK QUEUE	Average number of processes waiting for network transactions to complete
USERS	Average number of logged on users
PROCESSES	Average number of processes
ACTIVE PROCE	Number of processes using CPU time

Note: the following metrics are available on a per-application basis:

TRANSACTIONS	Terminal transaction count since last update
FIRST RESP	Average first response time for terminal transactions
RESPONSE	Average time to prompt for terminal transactions

Figure 3.

The performance of many intelligent network devices can also be monitored. These devices typically hold their metrics in a Management Information Base (MIB), and access is provided through the Simple Network Management Protocol (SNMP). [ROSE]

Summary

Management-by-exception techniques can be applied to performance management, and will reduce the time and expertise required to effectively manage the performance of your environment. This allows a more proactive approach to the managing your existing environment, and should reduce the time spent in problem resolution which will allow strategic capacity planning to be undertaken.

Performance management applications based on management-by-exception techniques are becoming available, and some are integrated with other network and system management facilities to enable an enterprise-wide "mission control" center approach to computer facilities management. When selecting or building an enterprise performance management solution, ensure it is both extensible to monitor all hardware platforms in your environment and open to allow custom analysis of the collected performance information.

Implementing management by exception requires you to determine your desired state, which can be best accomplished by defining your service level objectives. In some cases, measuring against these objectives may not be simple, and may require you to instrument application code, or to identify correlations between resource utilizations and user-perceived service levels. The thresholds used to determine exception conditions should be specific to your computing environment, applications, workloads and business priorities.

The concept of management-by-exception can be taken further to identify potential future problems as well as current performance problems. This provides a proactive approach to performance management , and will help ensure problems are identified and corrected before user productivity is affected.

The basic management-by-exception steps can be outlined as:

- Define your critical applications, transactions and batch jobs
- Establish your service level objectives for each of these
- Establish performance metrics to be monitored, including any code instrumentation
- If service levels can't be monitored directly, correlate to resource utilizations
- Specify thresholds for service levels and resource utilizations
- Determine exception condition severities based on these thresholds
- Implement an exceptions-based performance management application
- Monitor performance on an exceptions basis
- When exception conditions are identified, isolate according to priority and severity
- When condition is isolated to particular node, use detailed node-specific diagnostic tool to identify the resource bottleneck and affected processes (and clients for client-server applications)
- Take corrective action according to the application's importance to the business
- Note over-utilized and under-utilized resources for later examination during capacity planning

* UNIX is a registered trademark of UNIX System Laboratories Inc. in the U.S.A. and other countries.

References

[McBRIDE] McBride, Doug, *"Service Level Agreements: A Path to Effective System Management"*, HP Professional, August 1990

[MILLER] Miller, George W. (Bill), *"Service Level Agreements: Good Fences Make Good Neighbors"*, Conference Proceedings of the Computer Measurement Group (December 1987), pp. 553-557

[MORRIS] Morris, Wayne S., *"Performance Management in a Distributed Computing Environment"*, Conference Proceedings of the Computer Measurement Group (December 1991), pp. 620-625

[ROSE] Rose, Marshall T., *"The Simple Book An Introduction to Management of TCP/IP based Internets"*, Prentice Hall, 1991

Quantifying Investments in Technology

Mike Overly, Neal Elgersma
Hewlett-Packard
210 West 103rd St.
Suite 100
Indianapolis, IN 46290
(317) 844-4100

The issues related to companies realizing benefits derived through implementation of information technology means quantifying investments in technology. Benefits can be in three different categories. First, there are the benefits with direct impact on costs or revenues. Some examples are; Future cost avoidance, Lower personnel costs, and Lower material costs. Next there are benefits which have a major impact on a companies' effectiveness. Some examples are; Improved purchase order accuracy, Increased sales force productivity, and Improved customer service levels. Finally, there are the intangible benefits which directly support a company's goals. Some examples are; Improved quality of work, Higher employee morale, and other intangible competitive advantages.

Many companies have an information technology plan that includes a list of projects that will be done over a period of time, typically 1 year. Often, this project list comes from adhoc efforts from a number of different sources. Top executives pet projects, and formal user survey are two examples of possible sources of the projects.

Most top executives would agree that having the correct data and effective access to it is critical to the long term success of their companies. With such an importance it probably makes sense to devote some time developing or implementing a method to ensure that a company is receiving the best possible pay back from their investments in technology.

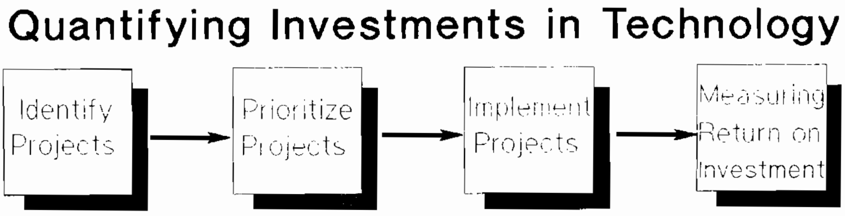
Many executives are hesitant to invest in technology systems without an analysis of its' pay back to the company. The solution is to quantify the benefits to the attainment of the strategic goals of the company.

For today's companies, increased competition has caused the need for rapid responses to changing conditions. In this environment, having the correct information technology solutions implemented will make a financial difference for some and for others may represent an opportunity to survive.

Certain assumptions are made when determining which investments to make. Today, in many companies, investments in technology are being evaluated and compared to other investment opportunities. For example, a $100,000 computer acquisition should provide a favorable return on investment relative to other investment choices. It is in many ways no different than the options that individuals have when they make personal investing decisions. Should I put my $1,000 in a bank savings account, money market, bond, stock market, mutual funds, etc. The individual must compare the expected return on investment of each of the investment choices. In addition, the individual might match the objectives of the investment with his/her own personal investment objectives. What is the risk? How long will my money be tied up? This process that each individual goes through is very similar to the decision process that an increasing number of companies must perform when considering investments in technology.

Lee Iacocca was one of the first top executives to start asking the tough questions like; What will this technology do to help my business? Many of the technology investments that Mr. Iacocca made were justified by reduction in head count, reduced operating expenses, etc. In his book, **Talking Straight**, he talked about how he was able to get his people to start quantifying the expected return on the company's dollars when considering investments in technology. Then it got to the point that he had signed so many projects, there should have been nobody left. He then had his people check to see if the expected savings were actually realized as a result of the company's investment in technology.

There are four basic steps to the process of quantifying investments in technology:


Quantifying Investments in Technology
Identify Projects
Prioritize Projects
Implement Projects
Measuring Return on Investment

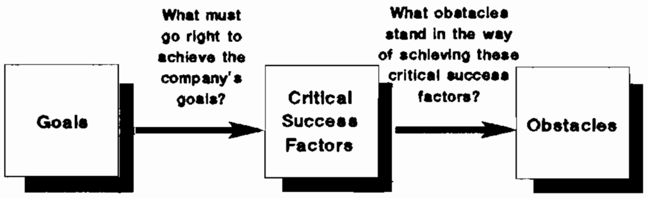
We will take each of the above steps and describe the process of accomplishing each.

THE PROCESS OF IDENTIFYING PROJECTS

The objective of this step is to identify all the projects that will help a company achieve its' goals and critical success factors. This includes projects that will either prevent or help to overcome the company's potential obstacles.

There are three basic steps to the process of identifying projects:

Identifying Projects


Goals
What must go right to achieve the company's goals?
Critical Success Factors
What obstacles stand in the way of achieving these critical success factors?
Obstacles

A company's goals are those few things that the company needs to accomplish each year to be able to stay in existence. The critical success factors are those things that a company must do to achieve its' goals. An obstacle is something that may prevent an organization from achieving its' goals and/or critical success factors. For example, one of a company's goals may be to increase profitability. A corresponding critical success factor may be to reduce or possibly eliminate pricing errors. An obstacle to this may be lack of timely management information related to the market price.

Goals should be formally documented and reviewed once a year. There may only be some minor fine tuning to the goals once they are documented. To achieve the goals, critical success factors and obstacles, start with your organization's top executive and his/her key executives. Individual meetings should be set up with each executive. The meetings should be short, approximately one-half hour, with the objective of documenting the above. This discussion will logically lead to some ideas on how technology can be applied. As much quantification of how the potential project could save or profit the organization should also be gathered here. This data will be used in the next step. Some sample questions that may be used are:

Goal Questions:

* What are your goals?
* What is your timeframe for achieving these goals?
* How would you quantify these goals?
* Can you prioritize these goals?

Critical Success Factor Questions:

* What things have to go right for you to achieve those goals?
* Can you quantify these?
* Can you prioritize these?
* How do you define success?

Obstacle Questions:

* What obstacles are currently standing in the way of the things you said have to go right?
* Are there inhibitors to ...?
* How could ... be improved?
* What changes would you like to see to deal with the inability to ...?

Next, group meetings should be set up with small groups of key users of areas that potential projects may impact. The objective of these few meetings is to verify the information already collected from the executives, gather more detailed information and to possibly identify other related projects that have not yet been identified.

The process of identifying projects varies widely from company to company. Some companies have a very formal procedure to identify and document technology projects. Other companies prefer to keep the process much less formal. It's most common to find a company that uses a combination of both formal and informal procedures.

An example of a company utilizing both formal and informal procedures to identify projects can be found in a typical Fortune 500 company. This company interviews functional managers once a year to ascertain what projects the managers would like to see completed in the next year. Then, once a quarter the status of the projects is reviewed with the functional managers. In addition, throughout the year high ranking executives give the information technology department some other "pet" projects that they would like to see completed. Finally, a particularly vocal order processing manager, who is always complaining to the IT Department, gets a project on the list to be completed. One can see that this particular Fortune 500 company has many ways to identify projects.

In summary, projects will be identified in meetings with top executives and his/her key executives where goals, critical success factors and obstacles to these are identified. Potential technology projects will be identified as opportunities to either achieve critical success factors or overcome obstacles. This list will then be taken forward and reviewed with key user personnel in the organization to uncover additional information as well as other projects that have not yet been identified. These will then be documented and taken forward to the next step of quantifying investments in technology.

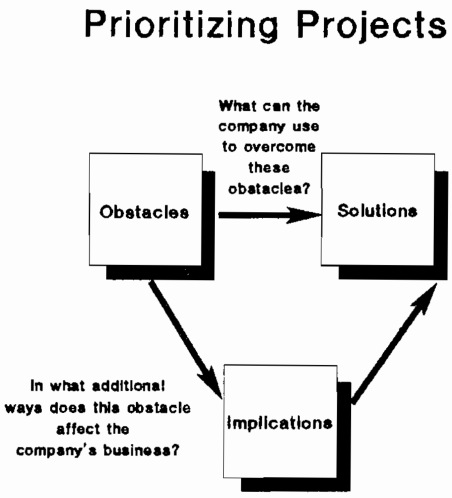
THE PROCESS OF PRIORITIZING PROJECTS

The objective of this step is to select those projects, from the list of projects which have been identified, that directly relate to helping a company achieve its' goals and critical success factors. The prioritizing of projects may seem both natural and trivial to be addressing. However, this step is not only extremely important to quantifying a company's investments, but unfortunately it is often overlooked.

Although the information technology department and functional groups within a company inherently realize the need to prioritize, they may not have all the data to do it accurately. They may also fall into the trap of being totally reactive to the most recent request for the next project.

In identifying projects the company's management team determined the goals, critical success factors and obstacles. It is important that this data along with the underlying vision of the corporation be clearly communicated throughout the organization. This process of "sharing the vision" allows all groups access to the data which eventually substantiates projects prioritization.

There are three steps to prioritizing projects:


Prioritizing Projects
What can the company use to overcome these obstacles?
Obstacles
Solutions
In what additional ways does this obstacle affect the company's business?
Implications

Recall that during the critical success factor meetings as much data as possible was collected to help quantify the obstacles. This data was then verified with the key users in small groups. These key users will also be required to identify implications of the obstacles. The implications are other ways that obstacles affect the company's business and should also be quantified by the users in terms of cost or potential benefit to the company.

The definition of these break-through opportunities lead to the correct prioritizing and help to develop the metrics by which an investment will be measured. Common questions asked during this phase include:

* How much money will this save the company?
* What will be the productivity gains for the department?
* How does that productivity gain effect the bottom line?
* How much will customer satisfaction increase?
* What is an acceptable rate of change or increase?

Attaching verifiable and quantifiable measures to projects with answers to these types of questions places a company on the path to correctly selecting the break-through opportunities. This also provides the basis for ultimately measuring the return on investment in the projects after implementation.

In summarizing the process of prioritizing projects, the following steps need to be taken:

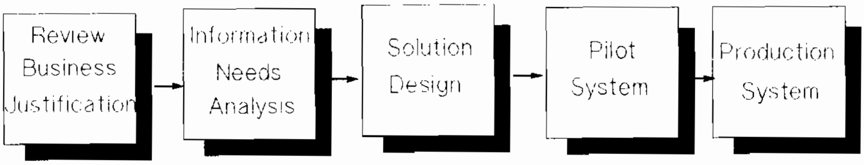
1) Acceptance of the need to effectively prioritize projects
2) Development of a corporate vision
3) Define the goals, critical success factors, obstacles, and implications to support the vision
4) Define break-through opportunities and quantify the expected return on investment
5) Prioritize the projects by recognizing which solutions leverage the greatest return, while contributing to the accomplishment of the vision

THE PROCESS OF IMPLEMENTING PROJECTS

Once a project has been identified and prioritized it is time to implement. A company would begin this step by selecting the project that was prioritized as having the best pay back to the company.

The process of implementing projects contains the following five steps:

Implementing Projects



This step, if done properly, not only addresses todays needs but future needs as well. There are a number of challenges to implementing projects and a number of alternatives.

Alternative 1 - Company provides all resources required for successful implementation. This includes technical, project management, etc.

Alternative 2 - Company outsources all requirements for a successful implementation.

Alternative 3 - Company does a combination of the above.

One of the most common causes of project slippage is that the project implementation team does not have the correct resources. Many companies feel they have the appropriate implementation expertise. Often, this is true. Just as often, those same companies who have the internal expertise are not organized in a way conducive to effective project implementation.

Sometimes, internal politics cause problems. When resources are needed from a different group within the organization and the managers of those groups never really feel committed to the project they may be hesitant to provide assistance. In addition, many times project managers are given all the responsibility and none of the authority. This means that when resources are needed to implement the project the project manager may not have the span of control required to get them committed.

Because of the potential implementation resource problems it is recommended that companies utilize a very special, focused implementation team that is composed of technology implementation experts. Just as other organizations have a SWAT (Strategic Weapons Assault Team) team that is called in on special, hard to solve issues, the same SWAT team approach is relevant for implementing technology projects. The strategic team would be armed with implementation weapons ready to assault the implementation.

The SWAT team should be as small as possible to achieve the decision making effectiveness inherent in small groups. In addition, the SWAT team should be exempt from internal political pressures that often lengthen the decision making process. A typical SWAT team might be composed of a project manager, two to three expert technical resources and two to three top user managers. The project manager should be skilled at strategic thinking and at gaining organizational support. Depending on the size and complexity of the project and/or organization there may be other resources actively involved outside the SWAT team.

First, the SWAT team should review the business justification that was utilized to identify and prioritize the project. The objective of this step is get the SWAT team up to speed, as quickly as possible on the background of the project and all the information already gathered. This can be accomplished effectively by assembling the entire team together in a conference room and reviewing all the data before leaving.

The next step is the Information Needs Analysis. The objective of this step is to assess and document the current process and evaluate and document future needs. The output of this step is a Needs Assessment Report. This document will not contain a product solution but instead will outline the requirements of the solution. This is typically done by a series of interviews of users and reviewing any existing process documentation.

The next step is the Solution Design. The objective of this step is to design the technical solution and typically includes the following:

* Solution selection criteria
* Solution design incorporating technical, functional and business requirements
* Objective comparison of solution alternatives
* Recommendation for a solution

The next step of Implementing Projects is the Pilot. The objective of this step is to produce a system that provides the "look and feel" user interfaces as well as the strategic system functionality. The 80-20 rule often applies to technology projects. That is to say that 20% of the functionality often provides 80% of the return on investment. The pilot system will lack some functionality but should be a real system, capable of cooperatively interacting with other existing systems.

At this point a partially functional, fully integrated pilot system exists. Once the users have received any required training they can begin using the pilot system. Any user interface or functionality change requests should be prioritized and assessed as to their cost and potential impact on the project schedule. After a reasonable pilot period it is time to incorporate the appropriate changes and move into the final implementation stage which is making the pilot a 100% production system.

Since the pilot is being used to do real work procedures need to be developed to test, and pass from development to production, the additional functionality. Some thought should be given to decide which functionality should be added first, second, etc., based on potential pay-back to the company. The pilot SWAT team should be phased out and replaced with production system integrators. This partially frees up your technology implementation specialists for the next highest priority project.

There is a direct relationship between the length of a project's duration and the risk associated with the project. In addition, for every week/month that the project slips, it will be one more additional week/month before the company realizes the benefits with which the project was originally justified. The SWAT team approach will maximize the speed of project implementation.

THE PROCESS OF MEASURING RETURN ON INVESTMENT

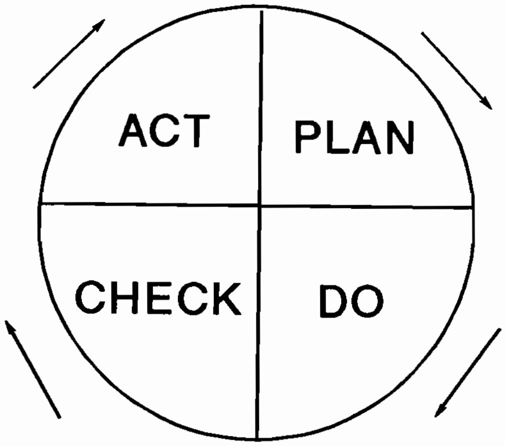
The most overlooked step in launching new information technology projects is measuring its' success. The hectic and demanding schedules of information systems professionals in most companies, coupled with the need to get on with the next project, generally leads to this oversight. As mentioned previously, Lee Iacocca recognized this as as issue and discusses it in his book **Talking Straight.**

It is absolutely necessary to grasp the concept of measuring projects and institutionalizing this philosophy within a corporation. We are actually referring to a topic much broader than return on investment (ROI) on a specific project or solution. The concept is continuous process improvement (CPI) and its' effect on project management.

Quite often projects are viewed as a finite set of steps to be taken to provide a solution to business needs. As discussed earlier, company goals and critical success factors must be reviewed regularly and updated to reflect current trends in a company's business. Just as these goals are updated, project plans must be updated. This concept is often referred to as the "living project" where feedback loops allow continuous process improvement to update solutions which continue to meet customers needs. Projects should be viewed as such a process.

The Deming Cycle or Plan-Do-Check-Act (PDCA) provides a pictorial framework for looking at projects in a new way.

PDCA - The Deming Cycle



The identification and prioritizing of projects can be associated with the "plan" quadrant of the Deming Cycle, while the implementation falls into the "do" category. The "check" and "act" is what is often missing and will focused on for quantifying investments.

When interviewing key company executives, questions should have been asked to help prioritize the project. An example of the break-through opportunity and basis for selecting it could be that a new order processing system would increase customer satisfaction. Assuming that this project has been completed and the order processing system was successfully implemented, an important question remains. Was the customer satisfaction increased as expected, therefore, giving a return on this investment?

To determine if the ROI was acceptable, you develop what is commonly called the issue statement. An issue statement indicates the intended result of the process intervention. It is developed from the customer information as described previously. The statement is composed of an indicator of change or direction, a quality indicator, and a reference to the process. Relating to our order processing example, a specific issue statement could have been "To increase customer satisfaction with order processing". This statement is still quite broad and further work must be done to have a verifiable and quantifiable issue to be tested in determining if there is a return on the investment of creating a new order processing system.

The next step is to review your process flow diagram. It is likely that this step was developed and documented either before or certainly during the project implementation. There are three common components of any process:

1) It can be drawn or diagrammed
2) Its' performance can be measured
3) It can be improved by discovering the relationship between its' components and its' performance

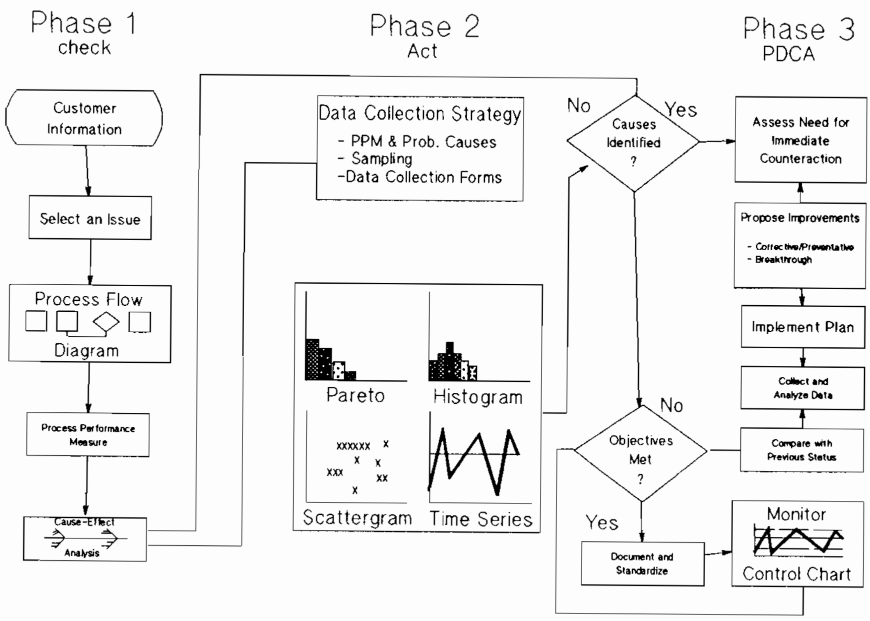
The pictorial representation of a process is the process flow diagram. If the diagram was not developed previously, it is important to do so now. If this is a fairly straight forward and simple process you can probably do it by hand. If the process is large and complex, like a corporate order processing system may be, there are proven methodologies to help define and map a process. The Hierarchical Process Modeling System and IDEF0 are a couple of examples of methodologies used for process definition and re-engineering of processes. The benefits of developing and using a process flow diagram are:

* It gives a picture of the process
* It assists in explaining the process
* It indicates relationships between steps in the process
* It can identify unnecessary complexity in the process
* It can identify which steps may impact process performance
* It can identify where to collect data for further investigation

Developing the correct metrics is fundamental to understanding and improving any process. The development of these metrics during the identifying and prioritizing phases gives the reference for accurately quantifying investments. You can now standardize these measures to give an objective evaluation of whether the process has actually been improved upon. These measures are called Process Performance Measures (PPM's). A PPM is defined as a measure of quality of a process and is composed of two parts. The first is a quality indicator, usually the same as in the issue statement. The second is a unit of measurement to quantify the quality of the process. To develop one PPM for our order processing example, determine the following:

* The quality indicator is: Customer satisfaction with order processing.
* Unit of Measure: Data entry errors from receipt of order to customer shipment.
* PPM: The number of data entry errors in processing an order.

PROCESS IMPROVEMENT MODEL



A firm measure has now been developed to determine process improvement. Information such as the current number of data entry errors should have been collected during the functional manager and key user interviews to establish a baseline reference. Key executive interviews may have yielded corporate goals such as a 5X improvement in customer satisfaction. Combining this information with your PPM gives the specific metric to determine the return on investment.

The last step falls into the "Act" quadrant of the Deming Cycle and requires the collection of data at process points to monitor the process improvement. If the results demonstrate that the PPM's were achieved and, therefore, the issue resolved, you have quantified the return on investment. This ties the information technology investment directly to accomplishing the corporate vision. Complete documentation and standardization of the successful process or project now must be accomplished. You have the basis for insuring that this project does not die, but by on-going monitoring of the process with tools such as the process flow diagram and control charts, you are on your way to continuous process improvement.

If the results demonstrate that the PPM's were not achieved, you have not necessarily failed. It simply means that more process refinement is required to reach the ultimate goal. In fact, quite often intermediate milestones are required to demonstrate progress against aggressive break-through objectives.

SUMMARY

In summary, one can see that the process of quantifying investments in technology involves all aspects related to technology projects. From the identification of potential projects, to the prioritization of those projects through the implementation and finally the measurement of the return on investment.

It may make sense to review the way your company handles investments in technology to see where improvements can be made. A Fortune 500 Information Systems Director recently stated; "I realize that change is inevitable. My job is to incorporate change better and faster than my competition." Hopefully, the process and some of the ideas discussed in this paper will provide the catalyst for some companies to incorporate change better and faster.

What's New & Improved with MPE V

By ROBERT ROSS

Hewlett Packard
Software Technology Division
Roseville, California
916-786-8000
National Interex Conference: August, 1992

Introduction

MPE V continues to be supported and enhanced. Many enhancements and quality improvements are available in Release 23 and Release 2P, scheduled to ship summer and fall of 1992, respectively. Customers can look forward to succeeding releases in the future that will also contain enhancements and fixes. This paper covers HP's current plans for MPE V, primarily focusing on Release 23, although the enhancements and fixes that have been implemented in Release 23 also apply to Release 2P.

What's Described in this Paper - MPE V Enhancements

Descriptions of the major enhancements for Release 23 listed below are available online through the MPE HELP subsystem by using the NEW keyword. The information displayed when you type HELP NEW is a short description of the enhancements and how to find more information for each. The following is a short description of the enhancements covered in this paper.

The MPE V EXPLAIN Utility - HP EXPLAIN allows users quick, online access to explanations of MPE V system error messages. It can run on any terminal or PC (using a terminal emulator), and does not need any special capabilities.

STORE/RESTORE Multiple Exclusion Filesets - STORE, RESTORE, and SYSDUMP file sets are more powerful. They now allow up to 9 exclusion file sets for each file set.

RESTORE Tape Creation Date Option - The tape creation date can now be displayed during a RESTORE with the new keyword TAPEDATE.

Improved RESTORE Performance for DDS - The MPE V DAT tape driver, STORE, and TurboSTORE now utilize the fast search capability of the DAT tape drives. The result is a significantly faster restore of individual files from a large DDS STORE or TurboSTORE tape.

SPOOK5 has a New DELETE Command - SPOOK5 now allows wildcard spool file deletes by date, priority and USER.ACCOUNT. Date and priority wildcards allow >, <, =, >=, <=, and <> operators. USER.ACCOUNT allows either USER or ACCOUNT name or @.

New Information From SHOWOUT - The SHOWOUT command now displays the completion percentage for active spool files.

System Console Status Release - The logical console is now returned to ldev 20 if a user logs off with the logical console assigned to their terminal ldev if the ldev is a modem port.

New Items Available Through the JOBINFO Intrinsic - The JOBINFO intrinsic now returns job initiation information.

Measurement Interface Expansion - The Measurement Interface will be expanded to support the maximum number of processes on MPE V (from 628 to 1024 processes).

FCOPY Now Copies ACDs - FCOPY will now copy by default the file ACD (Access Control Definition).

Introducing The Known Vendor Table - This new data structure will be available for third party developers to help them manage their data.

Job File Modification Date - As of Releases 23 and 2P, streaming a job will not cause the modification date in the job file label to be updated.

The New Dispatcher Priority Oscillation Boost Property - Enabling oscillation on a subqueue will cause the dispatcher to favor longer transactions and help prevent "long-transaction" type processes from being CPU "starved" when the system is very busy. The SHOWQ and TUNE commands have been updated to reflect this change.

Performance Improvement For COBOL Programs - Many large COBOL programs will show an increase in performance on Release 23 and 2P due to a change in the trap handling code on MPE V.

SUPPACCT.PUB.SYS Enhanced for Release 23 and 2P - FIELDJOB, extraneous error messages, and redundant commands have been eliminated, and the SYS account has been set to LOCK access=ANY to improve consistency between MPE V, MPE/iX, and HP Desk.

The following pages describe these changes in detail.

The HP EXPLAIN Utility

The new HP EXPLAIN utility will allow users quick and easy online access to an explanation of MPE V system error messages. These explanations may be displayed or printed, singly or in groups, or even using keyword searches. They consist of information about the cause of the error, plus an appropriate action (if any) that the user or system manager may take to resolve the problem.

There are three sets of messages that have been documented in this first release of HP EXPLAIN: The Command Interpreter (CIERR messages), The File System (FSERR messages), and the System Failure errors (SFERR messages). Additional sets of messages will be added in subsequent releases.

Syntax for the HP EXPLAIN Command

EXPLAIN.PUB.SYS may be run with the new MPE V Command Interpreter command, EXPLAIN. This command may be issued from a session or job. It may not be used from a program or in BREAK mode. Pressing [BREAK] during the command will suspend execution.

If you enter the EXPLAIN command without any parameters, EXPLAIN will continue to prompt you with the EXPLAIN> prompt for utility commands until you enter the EXIT command.

If you enter the EXPLAIN command with any of the parameters described below, EXPLAIN will execute the command, then return control to the MPE prompt.

```
EXPLAIN [message type and number]
        PRINT [message type and number]
        FIND [message number] [search string]
```

Parameters:

```
message type and number         Displays information about a certain
                                error message.  The message type and
                                number are the set or subsystem, such
                                as Command Interpreter (represented by
                                CIERR) and the message number (such
                                as 83).
```

PRINT [message type and number]	Prints a message and its explanation on the system line printer (class LP).
FIND [message number] [search string]	Displays all messages from the system catalog which contain the message number or search string specified.

A Sample EXPLAIN Command

If you would like to see the explanation for the following Command Interpreter error:

STREAM FILE MUST BE OF TYPE ASCII. (CIERR 83)

and then return control to MPE V, enter the following at the MPE prompt:

```
: EXPLAIN CIERR 83
```

Other commands within EXPLAIN are also available, such as

FIND	finds all messages matching text or number
PRINT	prints message and explanation to printer
TYPES	displays a list of message types
HELP	gives general information about HP EXPLAIN
?	shows your options at any prompt
EXIT	end HP EXPLAIN

STORE/RESTORE Multiple Exclusion Filesets

Both STORE/RESTORE and TurboStore have been enhanced to allow you to specify multiple exclusion filesets. A maximum of nine filesets may be specified as exclusion filesets to be "subtracted" from each fileset listed to be stored or restored. This feature is also available in indirect files. Although this functionality has been offered on MPE/iX TurboStore, it is not available when using the MPE/iX CMSTORE program.

Fileset exclusions may be used to save some resources (tape or time) for the system backup by excluding several accounts/groups/files which you do not need to store. Or it may be used to eliminate the indirect file which can currently be used in a "reverse" sort of way to exclude several unwanted filesets by specifying a large list of wanted filesets.

The current syntax for the STORE/RESTORE fileset specification is as follows:

```
STORE [{fileset] [,fileset]] [,...]}]
      [{!indirectfile              }]
              :
```

```
     RESTORE [restorefile]
             [;{fileset [,fileset] [,...]}]
             [;{!indirectfile          }]
                 :
```

The fileset parameter now has the form:

```
filesto(re)store[-filestoexclude[-filestoexclude[-...]]]
```

Both filesto(re)store and filestoexclude may be fully qualified in the form:

```
filename[.groupname[.accountname]]
```

The characters @, #, and ? can be used as wildcard characters in a filename, groupname, or accountname.

The following is an example of the usage of the new fileset exclusion feature:

```
     :STORE @.pub.sys-NMLG@.pub.sys-LOG@.pub.sys          ,&
     :      @.@.@-@.pub.sys-@.@.support-@.@.HP@-junk@.@.@ ,&
     :      *TAPE;SHOW
```

RESTORE Tape Creation Date Option

The RESTORE command has a new keyword, TAPEDATE, which displays the tape creation date. This enhancement has been added to both STORE and TurboSTORE so that every customer can use this feature. The tape creation date has always been placed on every STORE tape, so you can use this new feature with any STORE tape. Displaying the tape creation date is an excellent way to verify you have mounted the correct tape.

The new keyword, TAPEDATE, can be placed anywhere after the first two semicolons in the RESTORE command

```
     RESTORE [tapedevice];[filestorestore] [[;option] ...]

     option - one of the many optional RESTORE keyword
          options. See the online help text for all options.
          This enhancement adds the keyword TAPEDATE.
```

The date is displayed in the same format as returned by the FMTDATE intrinsic.

```
FRI, JAN 6, 1989, 7:39 AM
```

Sample use of the RESTORE Tape Creation Date Option

```
: RESTORE *tape;@.pub.sys;tapedate

   TurboSTORE HP30167A.00.11  (C) Hewlett-Packard Co., 1987.
   RESTORE *TAPE;@.PUB.SYS;TAPEDATE
   THU, NOV 23, 1972,  3:52 AM

   Tape Creation Date: THU, NOV 23, 1972,  2:39 AM
   WILL RESTORE      4 FILES;  NUMBER OF FILES ON TAPE =   4

   FILES RESTORED:         4
```

Improved RESTORE Performance for DDS

Restoring single files or small groups of files from DDS tape devices has been very time consuming. This enhancement utilizes the "fast search" capability of your DDS tape device and significantly reduces the time to restore single files or small groups of files. The performance improvement you see will vary, but in factory performance tests the restore for a single file at the end of a DDS cartridge took 48 minutes without "fast search" and 2 minutes with the "fast search" enhancement.

"Fast search" is completely transparent to the user and is supported both in RESTORE and TurboSTORE. There are no syntax changes and "fast search" is automatically enabled for most tape formats and hardware configurations.

Supported Tape Formats

"Fast search" is supported on all STORE tape formats except for Packed Labeled Tapes. Packed Labeled Tapes are created only when the STORE command contains the keyword PACKED and the tape device file equation referenced contains the keyword LABEL. Here is an example of how you create a Packed Label Tape format.

```
: file labtape;dev=tape;label
: store @.@.myacct;*labtape;packed
```

This example specified the file LABTAPE to be a labeled tape on device class TAPE. The PACKED keyword in the STORE command causes a Packed Labeled Tape format to generated. "Fast search" is not supported for this tape format.

Supported Hardware Configurations

All direct connect DDS tape devices support the new "fast search" capability. Remote DDS tape devices do not support "fast search". An example of a remote tape device specification is shown below. In this example "fast search" would be disabled.

```
file mytape;dev=remotenode#tape
restore *mytape;@.@.myfiles
```

This example requests the tape to be mounted on device class TAPE on the remote node named REMOTENODE.

Spook5 has a New DELETE Command

The Spook5 utility has a new command, DELETE, which allows the user to remove spool files in a way similar to the current PURGE command, but adds spool file wildcard options, DATE specification, and PRI specification. The spool file(s) can now be deleted by qualifying the READY date and the priority with relational operators (=, <, >, <=, >=, <>). The DATE is in a MM/DD/YY format, where MM is the integer month, DD is the integer day, and YY is the two-digit year (91 for 1991). The PRI option is an integer value from 1 to 13. Please note that if both the DATE and PRI options are given, both conditions must be satisfied for the spool file to be deleted.

SPOOK5 DELETE Command Syntax

```
DE[LETE] usac {; {D[ATE] <relop> mm/dd/yy } {, P[RI] <relop> output} }

usac        -->   user
                  user.account
                  @
                  user.@
                  @.account
                  @.@
                  *
                  dfid [, dfid] [,...]

<relop>     -->   valid relational operators (=, <, >, <=, >=, <>)

mm/dd/yy    -->   valid date in format of: mm (1-12), dd (1-31), yy (0-99)

output      -->   valid priority value (1 - 13)
```

Capabilities:

System Managers (SM) or System Supervisors (OP) can delete all spool file(s).

Account Managers (AM) can delete any spool file(s) in that logon account.

Standard Users (IA, BA) can delete any spool files that they created.

NOTE: The "usac" above follows the same syntax as the specification allowed on the existing Spook5 ALTER command. In addition, the capability checks for the DELETE command are consistent with the capability verification done with the ALTER command.

You should also be aware that because the DATE yy (year) parameter is limited to 2 digits, certain assumptions were made regarding yy values of the year 2000. The yy value of "72" is considered to be the birthyear of MPE, so any yy value less than "72" ("0 - 71") is considered to be the next century ("2000 - 2072"). A DATE value of "7/1/2" is July 1, 2002, and a DATE value of "5/1/74" is May 1, 1974.

SPOOK5 DELETE Command Examples

This first example will to delete all ready or locked spool files owned by the user MANAGER.SYS which also have a ready date of 12/2/91 or later.

```
> DELETE manager.sys; DATE >= 12/2/91
#FILE   #JOB    DEV/CL   SECTORS      OWNER             DATE     PR
#0408   #J93    LP            36      MANAGER.SYS       11/15/ 0  8
#0409   #J97    LP            36      MANAGER.SYS       11/17/71  8
#0446   #S27    LP            32      MANAGER.SYS       11/25/92  8
#0464   #S37    LP            32      MANAGER.SYS       12/ 2/91  8
#0471   #S42    LP            32      MANAGER.SYS       12/ 4/91  8
#0472   #S42    LP            48      MANAGER.SYS       12/ 4/91  8
>
```

This example will delete all ready or locked spool files under this account that have a spool file priority less than 8.

```
> DEL @; PRI < 8
#FILE   #JOB    DEV/CL   SECTORS      OWNER             DATE     PR
#0474   #S43    LP            32      MANAGER.SYS       12/13/91  4
>
```

This example will delete all ready or locked spool files owned by all users in all accounts that have a ready date less than January 1, 2000 (but greater than 1972, MPE's birth year) and that have a priority of less than or equal to 3. As the SHOW command displays, the files not deleted did not meet both conditions specified on the DELETE command (PRI).

```
> DEL @.@; DATE < 1/1/0, PRI <= 3
#FILE   #JOB    DEV/CL    SECTORS     OWNER              DATE     PR
#0480   #S44    LP             32     OPERATOR.SYS       12/13/91  3
#0487   #S48    LP             32     NOCAP.SYS          12/13/91  2
#0383   #J82    LP            108     ZUSER.QASYS        11/14/91  1
#0394   #J85    LP            108     ZUSER.QASYS        11/14/91  1
#0477   #S43    LP             32     MANAGER.SYS        12/13/91  1
#0478   #S43    LP             32     MANAGER.SYS        12/13/91  1
#0483   #S46    LP             32     MGR.TEST           12/13/91  1
> SHOW @.@;@
#FILE   #JOB    FNAME     STATE  DEV/CL    PR COP RFN OWNER
#0476   #S43    TEST      READY  LP        11   1     MANAGER.SYS
#09     #J4     $STDLIST  READY  LP         8   1     SMGR.TEST
#0475   #S43    TEST      READY  LP         8   1     MANAGER.SYS
#0484   #S46    AM        READY  LP         8   1     MGR.TEST
#FILE   LDEV    LABEL       SECTORS        LINES      TIME
#0476   %1      %2437513       32             4         9:23 12/13/91
#09     %1      %2471550       40            55       16: 7 11/18/91
#0475   %1      %2437453       32             3         9:23 12/13/91
#0484   %1      %2472020       32             3         9:42 12/13/91
>
```

New Information From SHOWOUT

The SHOWOUT command has been enhanced to make it easier to monitor the progress of files as they are printed. A new column, DONE, has been added to display the percentage of the file that has completed printing. That is, the percent that is shown is actually the percent of the file that has been sent to the printer. Therefore, if the printer has a large buffer the percent displayed will reflect what has been sent to the printer and may be larger than what has actually been printed by the printer.

With this new feature it will be easier to determine when the current file being printed will be finished. Situations such as trying to process a high priority print request or preparing to do a full backup will be made a lot easier by knowing approximately how much time is required to finish printing the current file. Also, you will be able to detect a paper out or paper jam situation on a remote printer more quickly by noticing that the percentage done on the current file being printed is not progressing.

Example of the new SHOWOUT report:

```
:showout
DEV/CL    DFID      JOBNUM  FNAME     STATE FRM SPACE RANK PRI #C DONE

LP        #04       #J2     $STDLIST OPENED      2048        8   1
LP        #0140     #J4     $STDLIST OPENED      2048        8   1
LP        #09982    #J257   $STDLIST READY         48      D 1   1
LP        #09983    #J259   $STDLIST READY         44      D 1   1
DUMPLP    #09881    #S'16   COVER    READY       2880      D 1   1
20        #09537    #S1268  $STDLIST OPENED
22        #09681    #S1276  $STDLIST OPENED
133       #010249   #S1349  $STDLIST OPENED
134       #010258   #S1352  $STDLIST OPENED
6         #05       #J16    $STDLIST ACTIVE      2048        8   1  %32

13 FILES:
    1 ACTIVE
    3 READY; INCLUDING 3 SPOOFLES, 3 DEFERRED
    10 OPENED; INCLUDING 6 SPOOFLES
    0 LOCKED; INCLUDING 0 SPOOFLES
    9 SPOOFLES:  15260 SECTORS
OUTFENCE = 6
```

System Console Status Release

With MPE V Release 23, any non-hardwired terminal which has been given the status of *system console* (by using the CONSOLE *ldev* command) will automatically release *system console* status when the user logs off or the terminal connection is lost.

Currently, any MPE V terminal not connected via DS can be assigned *system console* status, by virtue of the user of that terminal -- assuming they has sufficient capability to do so -- entering the CONSOLE *ldev* command.

A problem can arise if the user of that terminal later logs off the system without entering another *CONSOLE 20* command to revert the *system console* status. The problem arises when the user's terminal happens to be connected to the system via a modem or data-switch.

This type of situation most likely occurs at a customer site where central administration of multiple HP 3000s is practiced. At such multiple-HP 3000 sites, a data switch is often used for occasional connections required between the operations center and the remote HP 3000 -- employing the *CONSOLE ldev* command. In such instances, console logging messages such as job start/stop, session logon/logoff, device ready/not-ready, etc., normally print on the remote console; but, occasionally the remote console operator might inadvertently end the session by logging off or by terminating the connection without first entering a *CONSOLE 20* command. MPE V continues to attempt to print the console logging messages to the logical *system console*. Once the console logging I/Os begin encountering time-outs due to the connection being terminated, the result of storing these messages will exhaust MPE V's virtual memory space, resulting in a system failure.

With Release 23, the logical console is dependent on how the terminal with *system console* status was connected when that terminal logs off: If the logical console terminal is not hardwired, then the *system console* status reverts automatically to logical device 20 upon log off or connection termination (this means that *system console* status reverts to whatever physical system console *ldev* was detected when the system program INITIAL was executed). On the other hand, if the terminal that acquired system console status is hardwired, that terminal will retain system console status in spite of a log off by the remote console operator. The net result of this enhancement is to eliminate a common cause of system failures which is caused by neglecting to type a *CONSOLE 20* command before logging-off from a non-hardwired terminal that had been assigned *system console* status.

So now a hardwired terminal is the only case where a terminal can remain as the logical system console following the end of the session.

New Information Available from the JOBINFO Intrinsic

The JOBINFO intrinsic has been enhanced to provide access to eight new items, mostly pertaining to the initiator of a job or session. Much of this information is also provided by the Security Monitor product when a job is streamed.

When JOBINFO is called using new item numbers 41 through 47, the following information is returned about the user who initiated the job, rather than the job itself:

- Item 41 returns a DOUBLE value containing the Job or Session Number
- Item 42 returns a LOGICAL ARRAY containing the Job or Session Name
- Item 43 returns a LOGICAL ARRAY containing the User Name
- Item 44 returns a LOGICAL ARRAY containing the Account Name
- Item 45 returns a LOGICAL ARRAY containing the Logon LDEV #
- Item 46 returns a LOGICAL value containing the Job Initiation Date
- Item 47 returns a DOUBLE value containing the Job Initiation Time

The other new JOBINFO item number will let the caller find out whether a job or session is in QUIET mode, and pertains to the job/session itself, rather than the initiating job or session:

- Item 40 returns an INTEGER value defined as:
  0 = The job/session is NOT in Quiet mode
  1 = The job/session IS in Quiet mode

Calling JOBINFO with item numbers 37, 38 or 39 is not supported on MPE V.

Measurement Interface Expansion

The MPE V Measurement Interface has been expanded with MPE V Release 23 by increasing the maximum number of measurable processes, or pins, from the current 628 to MPE V's maximum configurable number of PCBs -- 1024.

More Measurable PINS - So What?

What does the ability to measure more pins mean to you as a system manager? The answer is with the Expanded MPE V Measurement Interface facility, you no longer need to lower the configured maximum number of pins simply to run your favorite performance tools -- GlancePlus/V or LaserRX/MPE -- along with your application load. This simple fact can mean great savings to you in hours of down-time, as well as in time saved you would otherwise have spent trying to duplicate the exact application load conditions your system was experiencing prior to the shut-down.

Should your system demonstrate unusual behavior (such as a slow-down) and you're not really sure of the cause for the behavior -- AND your system is one that needs every one of those 1024 pins because it is a heavily used system...then you can simply run one of HP's revised performance tools and determine what is causing the problem, right then and there, without having to wait for assistance.

Compatibility Issues?

With the introduction of the MPE V Measurement Interface Expansion, the number of measurable pins was increased, requiring neither a change to application programmatic interfaces nor to key data structures. Thus, existing performance tools will continue to execute as they do currently, even with the MPE V Measurement Interface Expansion installed.

The expansion was accomplished using two additional Extra Data Segments, and six previously unused pointer cells within the "MEASINFOTAB" Extra Data Segment. Those of you familiar with the current pointer cell approach will not be surprised by the fact that a similar approach was used for these newly added Extra Data Segments.

Although current versions of MPE V performance tools require re-configuring the maximum number of PCBs below 628, they will continue to operate as they do today, reporting correct information on the first 628 processes. Of course, installing the revised performance tools from HP will lift this restriction.

MI Expansion Details

Here are the details of the MPE V Measurement Interface Expansion for those of you that already know how the existing MPE V Measurement Interface works:

To start gathering process-level statistics, the performance application must first call the MPE V procedure STARTSTATISTICS with the parameter CLASSMASK set to the value 1 for process-level statistics, or the value %10 for miscellaneous process-level statistics.

Once STARTSTATISTICS has returned a condition code indicating no error to the performance application, the system will begin gathering its statistics according to the requested CLASSMASK parameter. The performance application may sample the collected data from time to time. This data sampling is done by copying information from the allocated Extra Data Segments into your program's data area (an explicitly allocated Extra Data Segment is recommended for this purpose). There are process-level counters maintained for every active process -- these are described in the MPE V Tables Manual. Also described there are the locations of the process-level Extra Data Segment and miscellaneous process-level Extra Data Segment within the MEASINFOTAB Extra Data Segment, number %73 - (59).

The process-level and miscellaneous process-level pointer cells currently consume three locations each within MEASINFOTAB:

```
MEASPROCXDSBANK  = MEASINFOTAB(%21) ... process counters, 1st XDS
MEASPROCXDSBASE  = MEASINFOTAB(%22)
MEASPROCXDSNUM   = MEASINFOTAB(%23)

MEASMPROCXDSBANK  = MEASINFOTAB(%32) ... misc counters, 1st XDS
MEASMPROCXDSBASE  = MEASINFOTAB(%33)
MEASMPROCXDSNUM   = MEASINFOTAB(%34)
```

and the newly-assigned cells occupy the following locations:

```
MEASMPROCXDS2BNK  = MEASINFOTAB(%40) ... misc counters, 2Nd XDS
MEASMPROCXDS2BSE  = MEASINFOTAB(%41)
MEASMPROCXDS2NUM  = MEASINFOTAB(%42)

MEASPROCXDS2BANK  = MEASINFOTAB(%43) ... process counters, 2Nd XDS
MEASPROCXDS2BASE  = MEASINFOTAB(%44)
MEASPROCXDS2NUM   = MEASINFOTAB(%45)
```

For pins 1-628, the counters are maintained in the first pair of Extra Data Segments -- both for process-level and miscellaneous process-level counters. For pins 629-1024, the two sets of counters are maintained in the second pair of Extra Data Segments.

All of the MPE V Measurement Interface's Extra Data Segments will be maintained as long as the application program that called STARTSTATISTICS is executing; the Extra Data Segments are de-allocated, and the counter modifications cease automatically upon the termination -- normal or abnormal -- of the originating performance application program. This behavior is unchanged between the current Measurement Interface and the Expanded Measurement Interface -- except now there are two additional segments subject to deallocation.

This information -- and much more -- is available to you in the new MPE V Release G.23 Tables Manual, part number HP 32033-90147. This product may be obtained through your HP Sales Representative.

FCOPY Now Copies ACDs

FCOPY has been enhanced so that copying access control definitions (ACDs) is the default of the product. You no longer need to specify COPYACD to transfer ACDs to the *tofile*. In addition, a new function, NOACD, has been added. Use this function when you do not want the ACDs in the *fromfile* copied to the *tofile*.

Either of the two examples below will copy existing ACDs to the *tofile*:

```
>FROM=ACDFILE;TO=NEWFILE;NEW
       or
>FROM=ACDFILE;TO=NEWFILE;NEW;COPYACD
```

The example below will not copy existing ACDs to the *tofile*:

```
>FROM=ACDFILE;TO=NEWFILE;NEW;NOACD
```

ACDs (access control definitions) are described in the MPE V Commands Reference Manual (32033-90006) under the ALTSEC command.

Introducing the Known Vendor Table

Beginning with Release 23 of MPE V, there will be third-party support of a new data structure called the "Known Vendor Table". This table will be stored in a data segment pointed to by a System Global Extension cell. It will be available on a global "first-come, first-served" basis for any vendor who would like to use it to store a word of information - usually a data segment number that the vendor has set up for sharing information among multiple users or processes.

Software developers do not have any standard way of storing and quickly finding global and/or local information for any or all of their users. The KVT allows any vendor to register and get an assigned number along with the code routines to access the KVT. This number would be an offset into the KVT data segment, and the routines would handle KVT initialization, vendor setup and takedown, locking, and reading and writing information into the vendor's buffer (data segment).

The vendor numbers will be assigned numerically as they are requested. To register for a vendor number, and find out how to get the routines to use the KVT, please contact:

Stan Sieler
Allegro Consultants, Inc.
2101 Woodside Road
Redwood City, CA 94062

Voice: (415) 369-2303
Fax: (415) 369-2304

Additional MPE V Release 23 Fixes and Enhancements

Some additional Release 23 enhancements and major fixes that will affect both system managers and program developers alike are described here as follows:

- Modification Date No Longer Updated When Job STREAMed
- The Dispatcher Now Recognizes EQ MIN/MAX Parameters Set Via The TUNE Command
- The New Dispatcher Priority Oscillation Boost Property
- SHOWQ Command Change
- TUNE Command Changes
- Performance Improvement For COBOL Programs
- Corrupt Message Files Will No Longer Fail The System
- MPE V Tables Manual Has Been Updated
- SUPPACCT.PUB.SYS Enhanced for Release 23

Modification Date No Longer Updated When Job STREAMed

A change was made in the STREAM command executor in V-Delta-4 to support the Security Monitor product. Unfortunately, this change introduced a side-effect that was felt by both customers who purchased the Security Monitor product and by those who did not. The side-effect was that beginning with V-Delta-4, the modification date in the file label of a job file would be updated when a job was streamed. The impact of this problem was felt in the following ways:

1) Partial backups got substantially bigger because job files would show up as modified, hence they would be backed up on partial dumps even though the files did not change.

2) Third party security products began reporting modifications to job files that were not actually happening.

This side-effect has been corrected in Release 23. Modification dates for job files will not be updated when a job is streamed regardless of whether or not the Security Monitor product is being used.

The Dispatcher Now Recognizes EQ MIN/MAX Parameters Set Via The TUNE Command

The MPE V dispatcher has never recognized the ES subqueue filter value and has always used the DS subqueue filter value for processes in both the DS and ES subqueues. This has presented a problem for customers who partially overlap the ES subqueue with either the CS or DS subqueues because there was no way to reduce the filter value for only those processes in the ES subqueue. The result would be processes in the ES subqueue having their priorities decayed much slower than processes in the CS subqueue because the default filter value for the DS subqueue is 1000ms while the CS subqueue filter value ranges between 0 and 300ms. If the subqueue priority ranges for the ES and CS subqueues overlapped, the impact of the ES subqueue processes on the response times of interactive CS subqueue processes was much greater than a system manager would expect. Customers who were aware that the ES subqueue filter value was ignored dropped the filter value on the DS subqueue knowing this would affect processes in the ES subqueue, but this would likewise affect the decay rates of processes executing in the DS subqueue.

On Release 23, the dispatcher now recognizes both the DS and ES subqueue filter values. Customers who reduce the DS subqueue filter value to speed up the rate of decay on ES subqueue process priorities should change their SYSSTART files, start-up UDCs, and/or start-up jobs to reduce the ES subqueue filter value via the TUNE command and leave the DS subqueue filter at the default.

The boost property can now be specified as either "decay" or "oscillate". The default boost property is decay and it is identical to the current boost algorithm. Processes begin their transactions at the base of their scheduling queue (CS, DS, or ES), and decay towards the limit of the scheduling queue as they consume CPU. Currently (and with the decay boost property), a process is reset to the base of the queue only when it completes a Dispatcher transaction. Dispatcher transactions are considered complete when the process blocks on a terminal read wait, a parent or child wait, pauses for more than one second, or indicates end of transaction in a call to the IOWAIT intrinsic.

Longer transactions (transactions that require more CPU to complete) will therefore decay to the limit of the scheduling queue and remain there. Given a large amount of activity at the base of a scheduling queue, processes at the limit of the queue will receive minimal CPU time and their transaction times will increase. For most production environments, this is desirable in that longer batch-type transactions have a minimal impact on the response times of processes performing shorter transactions. In some cases, however, a system manager may want longer transactions to get more CPU time at the expense of short transaction response time. It is this situation the oscillate boost property will address in providing a means of implementing this CPU scheduling policy.

When the oscillate boost property is specified via the TUNE command, processes are reset to the base of their scheduling queue once their priority decays to the limit of the queue. Processes will also be boosted to the base of the queue when they complete their transactions. A system manager can specify the oscillate boost property for each of the three user scheduling queues (CS, DS, and ES). The oscillate boost property allows longer transactions to be reset to the base of the queue so they can continue to compete for CPU and complete long transactions sooner. Specification of the oscillate boost property will prove helpful if processes of different transaction lengths are in the same scheduling queue and the shorter transactions receive excellent response time, while the longer transactions have much longer response times.

The oscillate boost property can also prove useful when scheduling queues are overlapped to allow batch processes to compete with interactive processes. Typically, when the CS and DS/ES queues are overlapped, it is done so the batch processes might compete with the longer transaction interactive processes. There are two distinct problems that may arise from this overlap. The longer transaction CS processes may suffer a great deal when they lose the CPU to the batch jobs at the top of the DS queue. It is also possible that the batch jobs do not receive much of a gain from the TUNEd overlap, because they quickly decay past the region of overlap and do not receive sufficient CPU. The oscillate boost property can be used to ensure the longer transaction CS processes do not remain at the bottom of the CS queue for an extended period of time, and can also be used in the DS/ES queue to ensure that the batch jobs are boosted back into the region of overlap. It may also be necessary to adjust the quantums (or filter values) of the queues so the boosting occurs with proper frequency. The quantums determine how rapidly the priority of a process decays. The priority of a process will decay more rapidly with a smaller quantum, and less rapidly with a larger quantum.

SHOWQ Command Changes

The SHOWQ command has been enhanced to report the boost property (OSCILLATE or DECAY) on each of the subqueue status lines. See the TUNE command and the Dispatcher change descriptions later for more information on boost properties.

```
:SHOWQ

DORMANT                                    RUNNING
Q  PIN   JOBNUM                            Q  PIN   JOBNUM
                                           C  M96   #S470

...

CQ MINQUANTUM=0, MAXQUANTUM=300, BASEPRI=152, LIMPRI=200, DECAY
DQ MINQUANTUM=1000, MAXQUANTUM=1000, BASEPRI=202, LIMPRI=238, DECAY
EQ MINQUANTUM=200, MAXQUANTUM=200, BASEPRI=180, LIMPRI=220, OSCILLATE
MINIMUM CLOCK CYCLE=1000
```

TUNE Command Changes

The TUNE command now supports oscillation for the three circular dispatcher subqueues. The TUNE command adjusts priorities, quantums and boost properties for the dispatcher circular queues.

OSCILLATE is a new boost property for Release 23. DECAY was the only boost property supported prior to Release 23, and is the default. Both boost properties cause a process's priority to decay from the base value to the limit. The rate of decay is inversely related to the quantum. Whenever a dispatcher transaction (e.g. a terminal read) completes, the process's priority is boosted back to the base value. DECAY allows a process's priority to remain at the limit until a transaction completes or the process is preempted. OSCILLATE causes the process's priority to jump to the base value as soon as the limit is reached.

TUNE COMMAND SYNTAX

```
                   {CQ}
TUNE minclockcycle;{DQ}                               {OSCILLATE}
                   {EQ}=[base][,[limit][,[min][,max][,[{DECAY    }]]]]]]
```

TUNE COMMAND EXAMPLE

```
:TUNE eq=180,220,,,oscillate
```

NOTE: Misuse of the TUNE command can significantly degrade system operating efficiency!

Performance Improvement for COBOL Programs

The Run Time Monitor (RTM) migration tool was introduced in V-MIT. This program required some changes in MPE's loader and abort processing code. The change in the abort processing code resulted in a significant performance degradation for large COBOL programs from the abort code having to check if prior stack markers were from the RTMSL segment. This check required a procedure call to loader routines that locked the LST SIR and traversed the LST Table looking for the RTMSL segment. This added CPU overhead as well as resource contention for the LST SIR. Because large COBOL programs (i.e. stack size > 16K words) frequently encounter arithmetic traps during address calculations, they slowed significantly after VMIT and increased system resource contention for the LST SIR and for the CPU. This overhead would be incurred regardless of whether or not the Run Time Monitor was enabled - the Run Time Monitor is only enabled while collecting data for an upcoming MPE/iX migration.

This problem has been alleviated in Release 23. The loader and the abort processing code now communicate whether or not RTM is enabled and if not, the abort processing code does not call the loader routine to search the LST for RTMSL segments. Before Release 23, the work-around for the above problem was to recompile large COBOL programs with the BIGSTACK compiler option which would use logical arithmetic for addressing rather than integer arithmetic and thus avoid the ABORT traps. The logical addressing, however, is slower, making this work-around less than ideal. Beginning with Release 23, COBOL programs recompiled with the BIGSTACK option to overcome the RTM performance problem should be recompiled without the BIGSTACK option.

Corrupt Message Files Will No Longer Fail The System

Beginning with Release 23, system failures 6601, 6602, and 6603 will no longer occur when the file system encounters a corrupt message file. Instead, the EOF is reset to zero when message file corruption is detected and an FSERR 105 is returned to the user.

MPE V/E Tables Manual Has Been Updated

The MPE V/E Tables Manual has been updated and expanded to document internal MPE data structures through Release 23. The Part Number for the Tables Manual is 32033-90147.

SUPPACCT.PUB.SYS Enhanced for Release 23

The stream job that the installer invokes to set the system's accounting structure defaults (SUPACCT.PUB.SYS) has been enhanced. With Release 23, FIELDJOB has been eliminated and the SYS account now has its default access rights explicitly set to allow LOCK access to ANY user. For any system that has had HPDESK installed, the SYS account has already been set up to provide this capability. This change should not have any adverse effects and will synchronize MPE V and MPE/iX in this regard. SUPACCT has also been cleaned up to eliminate redundant commands, and to correct a number of improper error conditions. This change reduces the output of this job by approximately one third for easier review.

MPE V/E Release Types

HP provides platform releases every 18 months. These releases are designed for maximum reliability, containing only problem fixes on top of the previous release functionality. Starting with Release 2P, platform releases will have a support life of 48 months, increased from 30 months, and receive future patches and retrofits. Platform releases are automatically distributed to everyone on a support contract.

Between platform releases, HP provides releases that include enhancements and fixes. These releases are available by request to customers on support contracts. The support life of these releases will extend three months past the next platform release or 12 months, whichever is longer. Release 23 falls into this category.

For customers currently on V-Delta-7, V-Delta-8, and V-Delta-9, it is easier to wait to update directly to Release 2P than to move to Release 1P first. The support life for these MITs have been extended to three months past the release of Release 2P so customers can remain on their current version and move directly to Release 2P upon availability, without the risk of remaining on an unsupported MIT.

Thanks go to Ralph Carpenter, Kathryn Jefferies, Yukihiko Umezawa, John Schimandle, Richard Santos, Dan Clavin, Steve Smead, Karen Lacey, and others at SWT for providing the material for this paper.

OSE, a framework for an Information Technology Strategy.

André Van Aken
Information Technology Consultant
Hewlett Packard Belgium
Woluwedal 100-102
B 1200 BRUSSELS
Belgium
tel. 32-2-761-3642

1. Introduction: Concepts of HP's Open Software Environment.

Two distinct trends, one business, the other technology related, are causing fundamental changes in the way forward thinking companies apply information systems. Changing market demands, competitive pressure, shortened business cycles, globalization and shrinking profit margins are forcing companies to be more flexible, more responsive to change.

At the same time the information technology evolves faster and faster. The price/performance improvements which the RISC architectures brought us, have been tremendous. Emerging new concepts such as Client/Server, Graphical User Interfaces and Expert Systems, make a redesign of the information infrastructure a necessity.

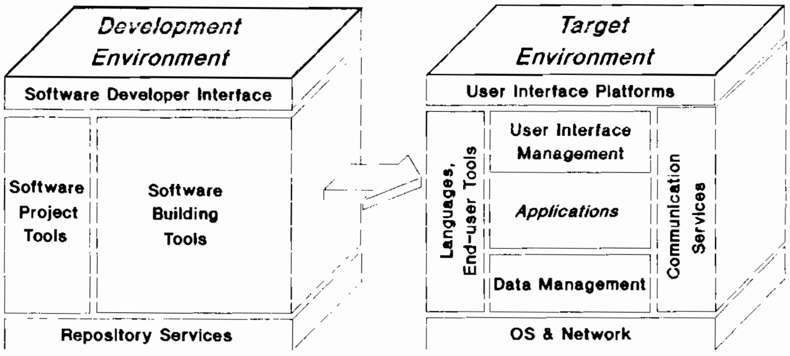
There is a need for information systems that support the strategic business functions, and that help the organization gaining a competitive advantage, by providing the tools to be responsive to changes.

Such an environment must be based on Open Systems, allowing the organization to apply information technology from many sources, and integrating these components in an easy and flexible way.

Open Software Environment (OSE) is Hewlett Packard's approach to define the way software should be developed in an open, multi-vendor world. It is, in a way speaking, a roadmap to build applications for Open Systems.

The core component of OSE is a reference model, the OSE framework.

OSE Framework



Applies to:
- Functional Area
- Application Type
- Current and Future

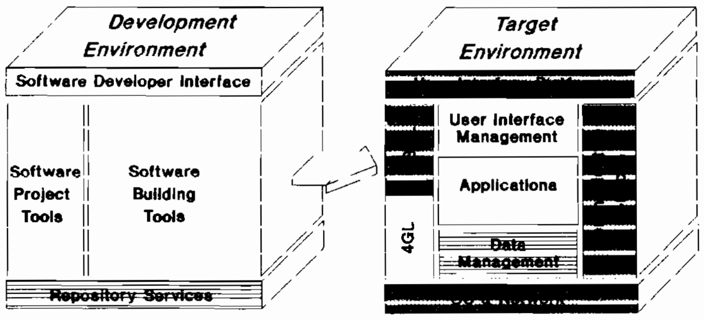
Figure 1

HEWLETT PACKARD

The OSE framework is a tool for analyzing and defining a software architecture. This framework is multi-dimensional:
The first dimension, shown in figure 1, shows the technology functional areas. To achieve the highest possible openness in the targeted production environment, these functional areas (e.g. data management, user interface management) must be carefully examined and filled in with appropriate products. Of great importance are the boundaries between the various functional areas, the call level interface, or API (application programming interfaces).

This is exactly the domain of standardization. The main task of standards committees is describing how subsystems must call each other. Where good standards are available (e.g. OSI communication protocols), each product that adheres to these standards will fit into the framework. Where standards are unavailable (e.g. UIMS) or are currently insufficient (e.g. SQL), the framework becomes even more important.

Open Software Environment
Levels of Standardization



■ Sufficiently standardized
▤ Partially standardized
□ Not standardized

Figure 2

HEWLETT PACKARD

Choosing a product in these functional areas involves a risk. Since no standard API's exist, products are not exchangeable. Isolating these components of the target application environment with an additional user-API is necessary, to avoid product lock-ins.

As a second dimension, we can look at the framework from the application type perspective. We can draw different frameworks (with slightly different functional areas) for transaction processing applications, business intelligence applications, or scientific and engineering applications. For instance, other types of languages will be used for the different application types,

Finally, we can project the framework into the future. Both the emerging technologies and the needs of the organization will evolve over time. We can think of time stamped frameworks, each time with other products filled in, as stepping stones towards an ultimate ideal framework, which will never be reached.

Adding the time dimension brings in new requirements; one example is the data management area: when we only consider today's state of affairs, data retrieval operations (e.g. the SELECT command in relational databases) are well standardized. There is no danger in coding database "SELECTs" throughout the application programs: we still will be able to exchange one RDBMS by another.

Looking forward, we see Object Oriented DBMSs appearing at the horizon. These will not only extend the functionality of database operations, but are also likely to be syntactically and semantically different. Therefore, a good strategy could be to isolate all database operations, including selections, to guarantee a migration path to these future technologies.

It is clear that the OSE framework is a powerful instrument for constructing an organization's Information Technology Strategy. Such a strategy must cover all aspects of information systems : deciding on which technology and products will be used in the software architecture, is not only the responsibility of developers. Information Technology has also an impact on the users of the resulting information systems, and not the least on the MIS organization itself.

To show how the OSE approach can contribute in the development of an IT Strategy, we shalll, in the following sections, explore two of the functional area's of the framework: **the data management area**, straight forward at a first glance, and the much more controversial area of **Graphical User Interfaces (GUIs) and User Interface Management Systems (UIMSs).**

3. Evaluating the data management area.

Processing and storing data is one of the most common reasons for organizations to use computers. This data is most often kept in databases. Computer programs, supporting the business processes, operate on this data, processing it in various ways (combine, analyse,...), in other words : data is used to produce information.

And whereas business processes, and by consequence also the application programs supporting them, are undergoing more and more changes, the data appears to be much more stable. The choice of a high quality database management system is therefore of strategic importance.

Many IT managers feel comfortable with modern relational database management systems. Nevertheless, it is important to take into account the OSE principles during the DBMS selection process. Let us examine some of the selection criteria for the various types of users: developers, end-users and Information Systems operational staff.

3.1 Developers.

A first obstacle to overcome when selecting a RDBMS, is the incompleteness of the currentl SQL standard. Whether only one, or multiple RDBMSs will be used, in either case the organization must decide on the way the databases will be accessed: will only standard SQL be allowed, for portability, complemented with user code, or does the organization try to benefit from the often powerful extensions that RDBMS vendors offer in their product? For example : most database vendors offer 'stored procedures' and 'triggers' in their product; however, the language in which these procedures have to be written. are all different, which decreases inter-database portability.

When selecting a RDBMS, an organization must also define the strategy for distributed data management. Important aspects are the communication protocols used for remote access, and the way incompatible SQL constructs are mapped into each other (e.g. the DATE data type can use a different representation in different RDBMSs). RDBMS vendors, who are committed to implementing the 'SQL Access Group' specifications, provide the best guarantee for openness and portability. Furthermore, not all database vendors provide comparable functionality for distributed database transactions (2-phase commit, locking, etc...).

For building strucurally sound, performant databases, a RDBMS must also be supported by the data modeling function of the existing (or to be selected) CASE tools. This selection criterion of course, is more related to the CASE tool itself, than to the RDBMS.

3.2. End Users

Database Management Systems usually provide easy-access facilities for end-users. When multiple heterogeneous RDBMSs are used within an organization, users must still be provided with a unique, consistent view of the information base. The selected database products must thus support the same application programming interfaces (APIs) for end-user access (e.g. the SQL/Access API), making it possible to provide an infrastructure such as IBM's Information Warehouse. Users must have the possibility to use various decision support tools against the databases, e.g. SAS Institute, Newwave Access, Lotus Datalens, etc...

3.3. Operational Considerations

The choice of a RDBMS has also an impact on the tasks of the computer operations department. To name a few: backup and recovery strategies, security, operator task automation, performance management, etc..., can all be influenced by the RDBMS. Even if developers design their applications in such a way that the database management system becomes interchangeable, incompatible operational utilities may inhibit this portability.

For example, if an online backup strategy is required and adopted, in order to allow 24 hour operations, it becomes difficult to migrate to a RDBMS that does not support this feature. Portability and inter-operability are also endangered if strategies for security, for recovery/restart, etc... are incompatible between database management systems.

4. Evaluating the User Interface

4.1. The Client-Server paradigm

Regardless of all benefits and issues for developers, end-users and computer operations staff, which we will discuss in the following sections, the decision for a graphical user interface (GUI) inevitably will require a Client-Server architecture. Driving the user dialogs entirely from a central server machine (as in the X-Windows topology) would not allow to achieve satisfactory performance levels, especially not in transaction processing environments.

Processing must thus be divided between a client system, which controls the user dialogs and presentation functions, and the server system(s) which manage(s) the database transactions. Client-Server applications imply a dramatic shift from traditional application environments, with consequences for developers, end-users and operations staff.

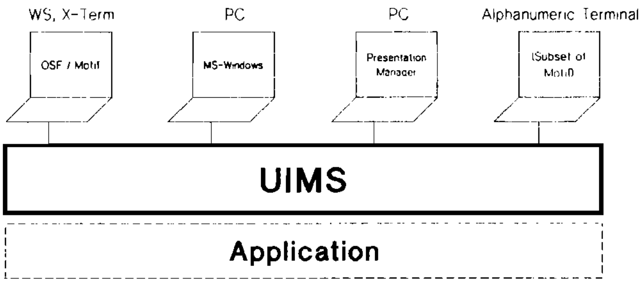
4.2. Developers

When implementing GUIs , the changes to which a developer must adapt are far more extensive than simply learning a new programming interface. The complete paradigm, i.e. the way the application interacts with the GUI product, will also change .
In the traditional application model, the application program calls the forms interface and manages the dialogs. With a graphical user interface, it is the GUI that calls the application : server functions are triggered by events taking place at the user level.

Three standards are emerging for GUIs : Windows 3.0 for MS DOS, Presentation Manager for OS2 and OSF/Motif for UNIX workstations. For designing user dialogs as well as the client components of an application, one must choose a development tool that is not restricted to only one particular platform. Ideally, the subsystem that supports the development and run-time execution of the user interface, should be client-platform independent, and if possible, also provide limited support for "dumb" terminals.

This architecture will offer all the benefits of the OSE framework : openness, portability and inter-operability. Care should be taken, however, in the design of a UIMS. Since no standards whatsoever exist in this domain, the UIMS-layer should be very well isolated from the rest of the application. Furthermore, it will not be easy to replace one UIMS product by another, for the same reason, namely lack of standards.
When applications are designed for graphical user interfaces, an organization must take a (calculated) risk when it makes a choice for a User Interface Management System. It has to balance the benefits of an intuitive user dialog, with the uncertainty of the protection of investments made in a particular UIMS product.

Features of a UIMS



- **Full Motif, MS-Windows, and PM Functionality**
- **Single, Easy-To-Program API for All Targets**
- **Separation of User Dialog from Application Logic**
- **Application Prototyping**
- **Client-Server Support**

Figure 3 HEWLETT PACKARD

4.3. End-Users

In general, users want a uniform, consistent view of all functions they normally perform: online transactions, business intelligence functions, electronic mail, appointment scheduling, etc.... While the broadest range of graphical applications exists for Windows 3.0, Presentation Manager and OSF/Motif provide similar, equally powerful functionality. Portability problems can arise when different tools and applications must interact at the user interface level. For example, for integrating a text processor in a transaction dialog in a Windows environment, one would typically use DDE-calls. This technique however would make the environment not-portable to, for instance, OSF/Motif.

4.4. Operational Considerations

Client-Server architectures bring new challenges to operations departments as well. The simple fact that many application functions are delegated to intelligent workstations (PC's or UNIX stations), causes new tasks and requirements to be generated. Software distribution suddenly becomes an issue : bug fixing, application enhancements, software updates, patches, are no longer well-controlled and well-defined actions, implemented on a central machine. Performing any of these activities on the client-platform means that the action must be replicated for each client system.

Management of software release and version changes, is only feasible if it can be automated. Tools for software distribution do exist. A standard in this domain, belonging to OSF/DME, is emerging , but not yet widespread. Until OSF/DME products will exist for the most important client- and server-platforms, it will be difficult to find one common, open solution for software distribution.
Similar issues arise in other operation areas such as backup strategy, security and performance management.

5. Using the OSE Framework for package selection.

From the previous chapters one might conclude that the OSE framework is only useful for building an Information Technology Strategy for in-house software development.

Of course it is true that software developers must select products and decide on tools and methods for all the functional areas of the framework. When buying packages, those decisions have already been made by the application builder, the I.S.V. (independent software vendor).

Nevertheless, in the Open Systems world, application packages usually come from different sources, often resulting in varying application architectures. It is important not to overlook these technological aspects in the selection process for an application.
From the OSE framework we learn that for packages inter-operability will be the most important 'open' quality. This co-existence is achieved on two battlefields : the database and the user interface, corresponding to the two areas we discussed before.

Different packages should at least support the same database paradigm; the relational database concept is still the most viable alternative today. Packages that can run as a separate layer on top of a RDBMS of the user's choice, provide even more openness : it becomes easier to exchange information between applications, to use the same tools for decision support, to provide a better logging and recovery of data, to optimize costs of licenses, etc...

In a similar way, the user interfaces must be compatible. Demanding that all applications must support a graphical user interface, may not be realistic today (it will be tomorrow however!). Minimum requirements for inter-operability must exist, e.g. the user dialogs must be able to run in a Windows based terminal emulator. Users must at least have the possibility of an easy context switch.

Selecting a package solely on the basis of functionality, can therefore be a dangerous decision. The hidden costs of incompatible technological constraints can outweigh the functional differentiators. The OSE framework provides excellent guidance to avoiding these errors.

6. Conclusion

Defining a strategy for Information Technology in an Open Systems environment, is not as straightforward as in the traditional proprietary world. Hardware and software technology are moving much faster than a few years ago. More and more CIOs and IS managers feel they are losing control over the technical aspects of their responsibilitiy area. Furthermore, they see their primary role in providing IT solutions that support their company's strategic business functions, and thus provide a competitive advantage.

For the technological areas of their IT strategy, they need guidance.

Hewlett Packard's Open Software Environment provides such a roadmap. It outlines a software architecture, as well as the means to fill it in with products, tools and methods. This approach will provide the basis for an Open Systems infrastructure, compliant with the principles of openness, portability and inter-operability, principles that are required to survive in the competitive world of the 90s.

Bibliography

Gray, Pamela. Open Systems: A Business Strategy for the 1990s.
London: McGraw-Hill, 1991.

Stonebraker,Michael. Future Trends in Database Systems.
IEEE Transactions on Knowledge and Data Engineering, Vol.1 No 1, March 1989.

Marcus, Aaron and Van Dam, Andries. User-Interface Developments for the Nineties.
IEEE Computer, Vol.24 No. 9, September 1991.

Hewlett Packard Company . Open Software Environment API Reference Guide

Hewlett Packard Company. A Roadmap to Open Systems : HP's Open Software Environment.
Version 1.0.

Paper Number: 3872

Managing Your Help Desk in the 1990s

Author: Tamara M. Gordon
Hewlett-Packard
100 Mayfield Avenue
Mountain View, CA 94043
(415) 691-5502

It is no secret that PC computing began as a grass roots movement throughout business organizations, nor that growth has been so explosive over the last decade that Information Technology (IT) departments have struggled to keep up. Now, PC networking is growing at a dizzying rate, connecting enterprise-wide resources, expanding the reach of the desktop, and redefining the meaning of "end-user computing". Trends are toward interconnectivity, heterogeneity, and distributed processing of more mission-critical applications. In simple terms, todays end-user computing environment is larger and more complex than ever before.

While some organizations have managed to rein in end-user computing under the guiding hand of IT, many are struggling to build support models that cost effectively meet end-user support needs in this complex environment. There has been a redefinition of support away from PC hardware and software support towards comprehensive user assistance. Old PC support groups have given way to a new concept, the help desk. The resources being devoted to this issue are evidenced by the recent growth in an industry group called "The Help Desk Institute". The Help Desk Institute is a membership organization for help desk managers that has more than doubled its corporate membership from the first year of operation in 1990 until 1992. Nearly 2000 corporations hold memberships with dozens of IT employees from each company participating.

This paper will examine the changing support needs of the end-user community, talk about some resulting trends in help desk management and then explore three alternatives for providing end-user support.

Support Needs for The Evolving End-User Computing Environment

In 1985 a PC support consultant from the Information Systems department might have taken a call from a PC user and heard, "Lotus 1-2-3 will not print my spreadsheet. Can you tell me what's wrong?" A help desk engineer might take a telephone call from the same user in 1995 and hear, "Lotus 1-2-3 for Windows will not print my spreadsheet. Can you tell me what's wrong?" It would seem from this scenario that not much has changed in 10 years. Not true.

In 1985 the user was most likely running a single application on a stand alone PC. He worked with only one data file at a time. And, his printer was probably connected directly to his PC. The PC support consultant would have explored the following potential sources of the user's printing problem:

User Error	Hardware Failure	Software Failure
1-2-3 usage	Personal Computer	Lotus 1-2-3
PC system configuration	Printer	Data file corruption
	Cable	
	Disk Drive	

By 1995 users will operate a far more complex environment. The desktop computer will be connected to a Local Area Network (LAN). The LAN will house applications that run in conjunction with software interfaces on the user's desktop computer. Data may be generated automatically on the LAN by the interaction of programs on different systems. For example, in our scenario perhaps financial reporting information is being generated by a program on the LAN that extracts contract information from an order entry database on a mainframe computer.

What potential problem sources must the Help Desk Engineer pursue now?

User Error	Hardware	Software	Network
1-2-3 usage	Desktop computer	Lotus 1-2-3 for Windows	LAN cabling
Desktop computer configuration	Network Servers	Microsoft Windows	Network routing devices
LAN client configuration	Mainframe computer	LAN client software	Network server configurations
Microsoft Windows configuration	Disk drives on desktop computer, network servers, or mainframe computer	Order entry software	
	Printer	Data retrieval software	
		Data file corruption on mainframe, network file server or desktop computer	

In short, as end-user computing has moved from isolated individual productivity tools to integrated organizational computing, the role of end-user support has expanded dramatically. Figure A displays how the end-user's "system" has changed from the 1980s to the 1990s.

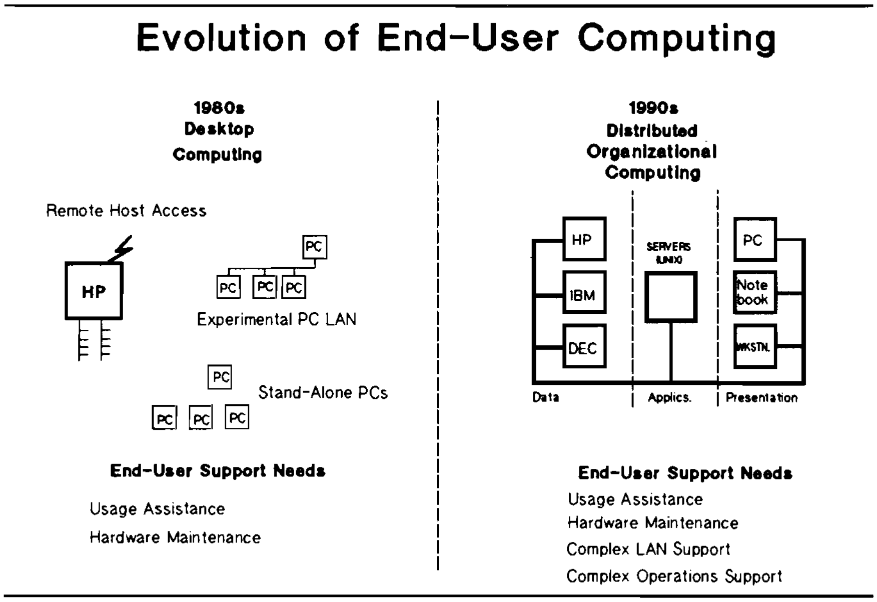


Figure A

Help Desk Management Trends

Information Technology managers are struggling to develop new models for end-user support in this complex computing environment. They are facing key issues regarding efficiency and effectiveness in delivery of end-user support, and they are asking themselves, "How can we cost effectively support end-users who rely on applications running transparently on numerous systems across a network that could span the building or even the world?" In response, three key trends can be seen in Help Desk management; integration, automation, and outsourcing.

Integration

As end-user support needs outgrew the PC support model of the 1980s, the first step in many organizations was to create a support group for "one stop shopping". Here users call for all their support needs, from telephony to PCs. According to the "Members Practices Survey" from the Help Desk Institute, approximately 75% of the members surveyed report that they plan to further expand the scope of their help desk operations in the next 5 years.

An integrated help desk becomes the end-user's lifeline to the entire support organization. However, that does not mean that the help desk becomes the entire support organization. Despite the increased breadth of help desk service, help desk engineers cannot be expected to become technical experts on all products and technologies. Instead, as network computing evolves in the organization, the help desk becomes a "foreground" layer in a multi-tier support organization.

In this way, the help desk serves as a valuable resource both for the end-user and for the rest of the support organization. According to the Help Desk Institute, approximately 80% of end-user inquiries are resolved in 15-20 minutes with the first call to an integrated help desk. The remaining 20% are escalated to the appropriate support resource, either within the company or to external support providers. Through this process, integrated help desks have been shown to increase user satisfaction at the same time that they conserve valuable IT resources for other projects.

Automation

At the first annual International Help Desk Conference in 1990, a handful of Help Desk management tools were displayed by software vendors. By 1992, dozens of software and hardware products, ranging from expert databases to voice response systems have entered the market. These products offer a wide range of help desk management functionality including:

Telephony management	Electronic Mail Access
Call Assignment	Network Access
Call logging	Vendor/Dealer Management
Caller Verification	Work Order Management
Workload management	Security
Knowledge Databases	Staff Training Management
Inventory Information Databases	Staff Schedule Management
Activity Reporting	Headsets

Tools and process technologies for help desk management have abounded because of the significant benefits that can be gained from automation. First, customer satisfaction is at stake. Effective call management is essential to the satisfactory resolution of a user's problem. Users who call the help desk are already in a state of discontent because they have a problem. Nothing will ensure the failure of a help desk like poor call management.

The second benefit offered by technology is data capture. Activity data is essential to ensure continued effective management of the help desk. For example, call logging tools typically capture data including peak call hours, average length of calls, types of calls, and resolution. This data can be used by the help desk manager to make critical staffing and training decisions. Further, help desk data can make a significant contribution to the effective management of the end-user computing environment throughout the organization. Problems can be tracked to determine needs for user training, equipment purchases, network performance optimization, and a host of other operational improvements to the distributed computing environment.

A third important tool set is knowledge database technology. Help desk engineers cannot be expected to possess both depth and breadth of technology expertise. Knowledge databases provide depth, lessen the rigor of training requirements, and allow knowledge and experience to be shared throughout the help desk support team.

Outsourcing

The third key trend in help desk management is outsourcing. According to a "User Wants and Needs" survey conducted by Ledgeway/Dataquest in 1991, companies expect to spend over 51% of their total PC software budget on support services by 1993. Over 30% of those support dollars are expected to flow outside the company for external or "outsourced" support. In fact, companies are expected to spend over one billion dollars on outsourced help desk support each year by 1993. The market research firm The Yankee Group predicts that 20% of Fortune 500 companies will ultimately adopt some form of outsourcing, while a full 100% will at least evaluate the outsourcing options.

Managing Your Help Desk in the 1990s

As end-user computing has evolved, there has been experimentation with a number of support management models. Let's explore 3 end-user support alternatives and discuss the benefits and costs of each. First, as a hold over from the 1980s, many companies have departmental support resources who provide direct support to end-users and rely upon an IT help desk function for backup support. Second, many companies have established multi-tier help desk operations that provide on-site response to end-user inquiries. And finally, an increasing number of companies are outsourcing help desk support to a third party.

Workgroup Champions

As PCs proliferated in the 1980s, many Information Technology departments recognized that they would be unable to keep pace with the demand for end-user assistance. A popular model for support was to assign a local guru, or champion, within each department or logical workgroup. It was the role of the workgroup champion to provide "first responder" support. Problems they could not resolve locally would be referred either to a PC support group or another group within IT.

Although this model was popular as PC computing exploded throughout organizations, the hidden cost of support is tremendous! The role of the departmental support liaison is often not formally recognized and is seldom included in an organization's calculations for the cost of support. Yet, according to a study conducted within Hewlett-Packard, the workgroup champion function can require as much as 40 person-hours per month for a department of 30 end-users. Further, an organization of 1000 end-users might have in excess of 30 workgroup champions.

The cost to a company of using departmental support should be explicitly recognized as:

- Time spent by the workgroup champion on support activities
- Time spent by the workgroup champion on training, if any formal training is offered
- Opportunity cost in lost productivity of the workgroup champion
- Opportunity cost in lost productivity of end-users for problems that could be resolved quickly by an expert
- Information Technology resources required for backup support to workgroup champions and end-users.

Centralized Help Desk Support

A centralized help desk function can provide a number of benefits, not the least of which is improved relations between IT and the end-user community. User productivity is improved through access to experts who can help them solve technical problems quickly. And, user training programs to encourage more effective use of technology tools can be designed based upon trends found in help desk call tracking data. The help desk can also provide efficiency in managing the end-user computing environment. Help desk call logging systems provide an invaluable mechanism with which to record the activity and problems on the distributed network and proactively identify opportunities for improvement.

Perhaps the easiest benefit to measure is the cost reduction that centralized help desk management can provide to IT departments. By providing an intelligent screen for end-user support, the help desk can significantly improve the productivity of other groups in IT. For example, one company estimates that installing a help desk function saved 30 minutes per day of each software developer's time, resulting in net savings of nearly $200,000 per year.

While many companies have found centralized help desk functions enormously beneficial, they are not easy to manage. Help desk managers face a number of challenges, the most significant of which is staffing. The success of any help desk depends upon the people who staff it. IT managers advise, "Want your help desk to succeed? Find good, patient people, give them technical training, and pay them a decent wage." Yet, help desks continue to be plagued by high turnover, stress and burnout, and difficulty finding engineers with the appropriate combination of people and technical skills.

In addition to on-going staffing issues, help desk managers face cost control challenges. PC LAN growth is ballooning at over 35% per year and help desk call volumes are going up as IT budgets are coming down. Four key things that help desk managers can do for cost containment involve proactive management:

1. Train end-users. The goal should be to eliminate basic feature usage questions. Some estimates indicate that adequate end-user training on features could reduce help desk call volume by as much as 60%.

2. Institute a system for inventory management. Help desk engineers cannot effectively support systems if they don't know what's out there. Unfortunately, manual inventory management is cumbersome and ineffective, since users move often and trade equipment freely. As more desktop computers become networked, automated inventory management tools are becoming more prevalent and may provide a workable solution by enabling on-line inventory via network access.

3. Institute a system for software management. Standardization and version control are musts for reducing training requirements for help desk engineers. Users can be encouraged to follow standards by restricting or eliminating support for non-standard software.

4. Manage printers for supportability. Estimates are from 40% to 70% of all help desk calls involve printing. Locate printers near users for convenient access and use simple, descriptive names in printer configurations.

Even when end-user support is managed proactively, help desk support does not come inexpensively. Consider a large division of a Fortune 1000 company. There might be 2000 end-users. The expense of maintaining appropriate personnel and technology resources in the help desk can be quite high. For example, with continuous technology changes, on-going training is a requirement for help desk engineers. It can require as much as 6 weeks of training to bring a new engineer on line. And, each engineer typically receives an additional 4 weeks of training each year to stay abreast of new products and technologies.

The cost of staffing the help desk is another significant factor. According to the "1991 Help Desk Salary Survey", published by the Help Desk Institute, a help desk supporting 1000-5000 users will employ 5 engineers on average. The average annual salary in the U.S. and Canada for a help desk engineer is about $30,000. Help desk manager salaries average approximately $45,000 per year. Considering salaries, benefits, and other overhead, the Fortune 1000 company in our example might consume as much as $400,000 per year in staffing expenses.

The third significant cost that must be considered in maintaining an on-site help desk is technology. Start-up costs for the technology to operate a help desk function include desktop computers, networking equipment, software for problem duplication, and help desk management tools and technologies. All of this can cost in excess of $60,000 for a help desk with 5 engineers supporting a few thousand users of standard business applications.

One important cost control trend in help desk management is consolidation. The Help Desk Institutes sites a number of member companies that have consolidated numerous on-site help desk functions to 2 or 3 help desk centers. The cost advantages to consolidation arise out of economies of scale that can be garnered in technology, facilities, and even staffing. However, since the resources required for corporate-wide consolidation lie outside the control of any individual IT manager, consolidation may be difficult or even impossible unless there is high level corporate sponsorship.

Outsourced Help Desk

Another trend driven by the need for cost containment is outsourcing. Many companies are turning to outsourcing as an alternative model for managing end-user support. Outsourcing is seen by some IT managers as a way to manage costs when large-scale consolidation is not feasible. And, outsourcing may provide the most cost effective model available to small and medium-sized firms.

External software support services are nothing new. Most software vendors offer free telephone support for their applications. In recent years, there has also been an emergence of pay-as-you-go services that provide software support, charged on a per call basis. While these services can be quite effective in answering some kinds of questions, they are not help desks. In fact, they provide a kind of support much more akin to the needs in last decade for supporting isolated end-user productivity tools.

Full-service outsourced help desk support is taking a different form. Companies adopting this model are developing partnerships with their support vendor. The vendor becomes part of the functioning of the IT organization and in return provides the same benefits of a centralized help desk for a fraction of the cost. An effective help desk vendor will provide call screening and resolve the majority of user problems without impacting the IT department. In addition, the tremendous cost savings to the company is hard to overlook. Using our example of a Fortune 500 company with 2000 users of standard business applications, most help desk vendors would charge around $10 per user each month for unlimited calling privileges. The total cost of outsourced help desk operations would come to under $250,000 per year. That compares with $400,000 just for staffing an internal help desk, not including any training or technology investments.

When considering an outsourced help desk relationship, there are issues that IT managers should still be prepared to manage. First, they should assign someone to manage the partnership with the vendor. Regular activity and performance reports should be provided by the vendor so the success of the help desk operation can be evaluated. Often, it is advisable to perform a satisfaction survey before implementation of the outsourced help desk and then repeat the survey after a few months of operation and then at regular intervals, such as yearly. Regular account reviews should be conducted with the help desk vendor to adjust the service offering as appropriate.

Secondly, the IT department may be called upon to maintain an updated user directory for the vendor so a useful level of activity reporting can be provided. For example, activity reports can often be broken down by location, department, or even by specific user if the vendor's caller database is maintained with the appropriate information. The importance of these activity reports should not be underestimated. They can be used to understand the performance of the entire end-user computing environment and make decisions regarding changes to it. The reports may prove valuable for prioritization of IT projects as well as decision making about other outsourced services.

IT departments should also expect to work with the vendor to establish the cross-functional coordination that will be needed for an outsourced help desk to provide effective problem escalation. Specific procedures and contacts must be established for the transfer of calls to other support providers, inc!uding the internal IT staff. The success of the escalation process should be reviewed on a periodic basis.

Finally, when considering a outsourced help desk solution, careful consideration should be given to the capabilities of the vendor. Remember that for effective support of a networked end-user computing environment, comprehensive support is a must. Help desk vendors must offer more than PC software support. Their support offering must provide an integrated solution that works as a partnership with the IT department to provide multi-tier support. The vendor must provide coordination with the groups that provide backup support like hardware and network maintenance as well as network and systems operations. Lastly, the help desk vendor should provide the kind of activity reports that not only evaluate the success of the help desk, but also provide insight into the workings of the end-user computing environment.

There is no absolute right or wrong solution for end-user support. Ultimately, the choice depends upon the needs of the organization. The key to success is carefully considering the strategic and tactical issues associated with supporting and managing the end-user computing environment and carefully assessing the costs and benefits of each support model.

A USER's EXPERIENCE OF HP3000 & DTC IN A MULTIVENDOR ENVIRONMENT

Juha HAKALA
Automation Unit of
Finnish research Libraries
Teollisuuskatu 23
SF-00510 HELSINKI
FINLAND
E-mail tkay_jh@cc.helsinki.fi.

1. INTRODUCTION

This paper describes our experience in using HP's Datacommunication and Terminal Controller in a typical multivendor environment. Our goal is to share this experience with other companies that are looking for viable networking solutions to meet today's business objectives.

Our project involved implementing a networked library system that linked sites throughout Finland. The primary objective was to provide transparent end-user access to resources throughout a multivendor environment.

Ensuring the success of a project of this type requires a thorough understanding of end-user expectations and requirements. In this paper, we first discuss the objectives of the project and the way in which it developed, identifying key elements and success factors. The second part represents the results of tests and evaluations done during 1991 and explains our choice of equipment.

This paper is also an opportunity for us to give an overview of this, an important and ambitious project, whose main objective is to automate all Finnish academic libraries and create a well-functioning inter-University library network.

2. The LINNEA Network

2.1 Project Statement & Objectives

The primary benefit of library automation comes through cooperation. The requirements for successful cooperation are : coordination and common standards. These fundamentals have been the guiding principles in library automation planning within the academic library community in Finland.

In 1984, the Finnish Ministry of Education initiated an evaluation process to research the best strategy for future library automation. As a result of intensive study the following objectives were outlined in 1986 :

a) An integrated library system should be installed locally for each university, each with the same software.
b) A central system should be established that provides services to the local systems.
c) Datacommunication networks and software should be used to link the local systems and the central system together.

Because the benefits of a uniform library network using the same software and protocols appeared so promising, the Ministry of Education decided to finance the project centrally. The project was started in 1988 and the first system was alive in the end of the same year. Resources of the project - for computer/network hardware & software- are roughly 50 million FIM or about 11 million US dollars.

This project was managed by the Automation Unit of Research Libraries - known as **TKAY** - in the Ministry of Education. The name of the network resulting from the project is **LINNEA** (Library Information Network for Academic Libraries).

At the moment the project which was scheduled to end 1993, is nearing completion with 16 libraries installed. The project is ahead of schedule and has also kept its budget well. We wish to express our gratitude to HP for helping us to reach this result, which is not common for a large governmental automation project -at least not in Finland.

2.2 Key-players in the project

This project targets the twenty University libraries in Finland, and two other libraries : the National Repository Library and the Library of the Parliament. TKAY is directly responsible for strategic choices, coordination between universities, co-operation with foreign libraries, coordination of implementation projects and coordination of feedback to vendors. Note that all strategic and global decisions are taken by a steering committee on which university representatives have a majority.

2.3 Key-Elements of LINNEA

2.3.1 Element #1 : VTLS system

After thorough testing of systems which responded to our bids, the Virginia Tech Library System was chosen in 1988 as the library system for Finnish research libraries. VTLS Inc. is one of the key players in the international library system market, with an installed base of approximately 200 units throughout the world. This software provides all the major library functions such as search, acquisitions, cataloging and serials control.

The VTLS environment is based on HP3000/900 systems and uses the Image database system which has proved to be an efficient and trustworthy combination. One of VTLS's advantages was its capability to meet our stringent localization requirements concerning, for example, character-set, record structures and native language support. This kind of adaptability is essential to Scandinavia and other European countries .

In 1988, the networking facilities of the VTLS software were outstanding when compared to other library systems. It allowed transparent use of remote VTLS databases. A user could search and transfer records from other VTLS systems. These features were - and still are - based on HP protocols VT/3000 and RFA, which belong to the NS/3000 product suite.

Due to the large sums invested in R&D, VTLS is an evolving product. One of the most interesting areas of improvement is currently the implementation of ANSI/NISO protocol Z39.50 version 2 (1992). This standard defines an interface through which different database systems can communicate with each other, thus allowing search and retrieval of records. In the US, there are several implementation projects in process; first beta were released in 1991. In august 1992 there will be about 10 available implementations of Z39.50, and in the end of 1992 this number will rise to 15-20.

VTLS is currently building a Z39.50 client to PC and a server to HP3000. The beta versions of these will probably be available in the end of 1992.

2.3.2 Element #2 : LINNEA libraries

In 1990, there were about 1200 people working in the Finnish University libraries. The total number of students was 113.600 and the number of staff at institutions amounted to 21.000 . For every librarian, there are then 117 students. Finnish University libraries can be freely used by anybody. Consequently, the number of potential users is quite large. In an environment like this, computers have to be effectively utilized - otherwise the library system would be hopelessly ineffective.

Collections included 14.4 million items; yearly growth was 400.000 books. The budget for all academic libraries -not including costs to build the LINNEA network- in 1990 was 213 million FIM or about 40 million US dollars. Number of book loans was 30 million books, or 20 per student.

As of today, the size of the library databases varies between 5.000 and 500.000 bibliographic records. A large amount of material is not in machine readable format, but there are many retrospective conversion projects going on. Partly due to this, databases are growing fast. The planned central system database, which will hold all the data from LINNEA libraries, will in the beginning contain 2 million records, with a yearly growth of about 300.000 records.

2.3.3 Element #3 : FUNET network

The Finnish Universities and Research Network **FUNET** was established in 1984 and today connects twenty universities and more than twenty research institutes in a single logical multiprotocol network, which uses TCP/IP and DECnet protocols. FUNET is connected to other Scandinavian networks (Finland, Sweden, Norway, Denmark, Iceland), to European research networks and finally to Internet, world-wide network of networks.

FUNET constitutes the physical network via which information is transferred between local systems and from local systems to central system database and vice versa. FUNET also allows network users to access any LINNEA library system from anywhere within the Internet. Likewise, Finnish librarians can access all the 300 other library systems which are now available in the Internet from their terminal or PC.

This network is basically a set of leased or public lines interconnecting all university LANs via level-3 routers.

2.3.4 Element #4 : Users

Users can be divided in two main categories :

a) **Librarians** who manage the database and use almost all available database functions. One of their main tasks is the input of bibliographic and other data into the database.

Their requirements :

Workstation : Librarians need HP 8-bit terminals or PCs with an HP terminal emulator to get the required character-set. Access from other terminals can not be used because of their character-set limitations.
Access : Librarians need to access both local system and other remote systems (either central or not) for reference searching and copy cataloging functions.

b) **Students**, University staff (and other people) are the "final users" of the LINNEA library systems. They are permitted to make different kinds of searches (title, author, subject code, keyword) or to reserve books. A typical search session contains different kind of searches in which search terms can be truncated and/or combined with Boolean logic.

Their requirements :

Workstation : These users should be able to use ASCII 7-bits or 8-bits terminals (with VT100 emulation) or PCs connected to University LANs and equipped with a Telnet emulator like NCSA/Telnet. The access to library systems has to be as transparent as possible, independently of whether the system is located locally or remotely and ignoring the operating system and its commands.
This requirement is also valid when using LAN-based terminal servers : i.e. the fewer commands students have to know, the better it is.

In all libraries, systems are used in Finnish, in Swedish or in English. The number of users accessing the databases varies a lot depending on the University. The largest library system in LINNEA (Helsinki University with 35.000 students) has been configured for up to 250 concurrent users. The system runs on a 3000/967. The smallest libraries have around 20-30 active users.

3. LINNEA REQUIREMENTS

a) Universities are already using many systems from different vendors (IBM, DEC, SUN, ALLIANT, etc..) connected to local area networks. These systems run a broad range of applications for different departments within each University. Any new system or equipment must fit into the existing networks in order to protect the previous investment. Librarians have to be able to use other systems via Telnet from their HP terminals and PCs.

b) All university end-users must be able to access the local library system. These end-users can use different types of workstations : UNIX workstations, LAN-attached PCs, Macintoshes or PCs and terminals connected asynchronously to terminal servers (for instance 3COM CS terminal servers on the OULU site).

c) These end-users (researchers, students) must have access to their local library system, as well as other LINNEA systems via FUNET, in the most transparent manner possible, i.e. ignoring where the remote systems are located on the networks, and getting direct access to the VTLS application (end-users should not need to interact with the operating system) .

The three requirements stated above , dictate the use of standard networking protocols . We chose to base our network on

ARPA-TCP/IP services (FTP, Telnet) to insure connectivity with the multivendor LANs of the universities.

d) The end-user connectivity solution must be able to evolve and must minimize additional purchase costs. The number of users in the largest systems is over 10 times bigger than the amount of users in small sites. On the other hand, the usage of every system is expected to increase with time fairly rapidly, so easy upgradability of the computer itself and its Telnet capabilities are important.

e) Lastly, solutions must offer database-record transfer capabilities to copy records from one LINNEA database to another, (not possible with TCP/IP services) in order to speed up the input of bibliographic data and avoid duplication of work.

3.1 Architecture of a typical Library System

With the HP3000 systems, it is possible to meet our networking requirements by combining HP ARPA services and NS Services on a TCP/IP network. The VTLS software has a built-in feature which allows record transfer between systems based on NS/3000. Other vendors did not offer this kind of service when we selected our system.

LINNEA library system's architecture :

a) An HP3000/900 system with the Turbo-Image database management system
b) The VTLS library system software and database
c) The HP NS/3000 and ARPA services for connections within and outside LINNEA :
    - to ensure networking within LINNEA
    - to ensure access from remote sites via FUNET
    - to ensure access to remote sites reachable with TCP/IP protocols.

The specific HP3000/900 system model differs depending on the site. Todays's installed systems range from 3000/920's to 3000/967's. The central system computer (to be installed in October 1992) will be a 992/100 -one of the first installations of new HP3000 in Europe.

3.2 Future Requirements

TKAY and the universities are looking for solutions able to evolve gradually , step by step, while implementing state of the art technology.

3.2.1 Evolution of the data-base

In the future, ability to search and transfer bibliographic records based on Z39.50 protocol or on it's international version - ISO SR Protocol - will be an important feature in library systems. The fact that VTLS will build Z39.50 clients and servers gives them credibility in the library system marketplace. Our belief is that only the products with well-developed networking features will be successful in the future.

The development of software based on OSI protocols is made more difficult by the fact that presently these applications have to run over TCP/IP networks. Luckily there is a PD software package called **ISODE** (ISO Development Environment) which allows to run ISO transport services on top of the TCP protocol stack. ISODE is currently used for example to run X.400 applications over TCP networks. ISODE runs on several UNIX platforms including HP9000 systems. POSIX-compliant operating systems can be used too, and therefore we are looking forward to MPE/iX with excitement.

VTLS is also building a UNIX-version of it's product, using a relational database model. UNIX and relational DBMS will require more computing power, but on the other hand it can at least in principle give more alternatives when choosing between different UNIXes and between different hardware platforms. Degree of portability of the not-yet-existing code is of course an open question.

3.2.2 Evolution of protocols

When OSI services like FTAM and X400 can be used over FUNET and in HP3000 environments, the LINNEA network will probably move on to implement other OSI network services as well. The possibility of upgrading hardware and software gradually - avoiding big investments - is an important feature for us.

4. TELNET & THE HP3000 SYSTEM

In order to provide terminal connectivity over TCP/IP networks, all systems must support the Telnet protocol. HP3000 end-users must be able to access all other non-HP computers, and other networked workstations, including non-HP systems, workstations and PCs. All network nodes must have access to the HP3000 system.

4.1 Situation as of End 1989

In 1988, when the selection of HP3000 and VTLS was made, the HP3000 network offering didn't include the availability of Telnet or any other ARPA services. At that time, HP was "committed" to providing ARPA services but no product was available. Plans from HP were to deliver the first products by end of 1990.

These plans included the delivering of FTP and Telnet protocols. The Telnet implementation was based on the DTC, therefore quite different from other native solutions : the HP3000/900 is accessed via the DTC which converts Telnet to the HP3000 proprietary protocol. Telnet hosts are accessed directly from the DTC without involvement of the HP3000.

4.2 Options for the Future

Because of the non-availability of Telnet at that time, we had to make the decision regarding the networking part of the first library systems to be implemented :

Option #1 was to use DTCs as "classic" asynchronous multiplexers. In this case, the HP3000/900 would be connected to the network via a "back-to-back" solution with all DTC ports connected to a Telnet/TCP/IP terminal server port. Access from Telnet hosts or terminal servers would be done via IP addresses corresponding to a pool of DTC ports.

Option #2 was based on a two phase plan :

Phase 1 :
* Connect DTC ports to terminals and to some Telnet terminal server ports (3COM CS).
* Use the switching capability of the DTC (therefore managing the DTC with the HP OpenView DTC Manager SW from a PC workstation) to provide DTC terminals with access to HP3000/900 and to other DTC ports connected to 3COM CS ports.

Phase 2 :
* Test HP's Telnet solution as a "beta-site" prior to the official introduction of HP products.
* Take the decision to go or not with HP's Telnet solution.

4.3 Decision

We decided to choose option #2, based on the following data :

a) Phase 1 could be put in place immediately .
b) Phase 2 was a good opportunity for us to qualify HP's solution before having to make an any important decision for VTLS connectivity and multiplication of sites..
c) This plan required no additional cost compared to the final solution.

We trusted that HP could deliver the required products when we made this decision to implement option #2 and wait for final products. This decision didn't impact the LINNEA implementation schedule.

This plan applied to only one University. All other universities decided to wait for HP's final Telnet solution. At that time, many Universities would have preferred a native HP3000 solution, as being the common solution available on other hosts.

4.4 Option #2 timeframe

The timeframe was the following :

```
Dec 89          Nov 90              Mar. 91           May 91
|_______________|___________________|__________________|
(1)             (2)                 (3)                (4)

 <--------------><------------------------------------->
     Phase 1                     Phase 2
```

The following four points correspond to the timeline featured above :

(1) Acceptance of interim solution. Implementation started in Helsinki University.

(2) OULU University wanted to test & qualify the Telnet solution before general availability. It became the pilot site. OULU University became beta-site for the "Telnet to HP3000" capability the first. The solution was tested in an operational environment, including a test of the maximum number of concurrent sessions as an access test between ten non-HP platforms :

- ALLIANT FX/2800
- IBM Mainframe access via VM TCP/IP on 9121/350
- IBM RS6000
- IBM PS
- SUN workstations
- DEC VAX with Wollongong and CMU-TEK
- DEC VAX/Ultrix
- 3COM terminal servers

(3) OULU University became one of the few beta-site for the full Telnet solution including "DTC to Telnet hosts" plus new functionalities such as multi-session.
The DTC to Telnet capability was also tested with the above list of non-HP systems

(4) End of beta-sites. Decision to move to HP's solution based on DTC.

During the two beta-tests, end-users tested access to remote Library systems through FUNET.

During the beta test, a few minor problems were found which were corrected afterwards. The beta-test phase was a useful step which gave us a better understanding of HP's HP3000 solution.

From our point of view, there are two important limitations in this implementation :

a) The lack of printer support via Telnet (or more precisely the lack of LDEV # pre-assignment or capability to use nailed devices over Telnet).
b) The impossibility to "call" outgoing Telnet.

Because of this last limitation, you can not start a Telnet session from inside an application, you must leave the MPE/XL prompt and return to the DTC user interface. This makes communication with other library systems more difficult, and more complicated to explain to the application users.

The impossibility to STARTSESS Telnet sessions was not a severe limitation for us, as we have different kinds of users accessing the systems via Telnet.

4.5 From tests to "GO" decision

Following the tests described above, TKAY and Universities decided to continue with HP's solution. The other Universities benefited from OULU's experience. A few Universities that were initially skeptical, made their decision after seeing the success at OULU. It gave the reassurance that HP's solution satisfied the demanding requirements.

For sites where HP3000 systems and DTCs were already installed, existing DTC48s were upgraded in order to run the new release of HP OpenView DTC Manager. Lastly, the HP OpenView DTC Manager release 10.5 was installed . This hardware and software update was easy and not expensive. More important, it gave each site the flexibility in migrating from HP3000-dedicated DTCs to Telnet-DTCs either in one step or an multiple steps basis.

Most of the Universities - new installations - did not have to buy upgrades, but rather buy new DTC and Telnet Access products.

4.6 What we learned with these tests

The HP's HP3000 Telnet solution is a logical starting point for multivendor connectivity.

From the cost perspective, this solution is inexpensive if a site is already using an HP OpenView workstation to manage DTCs.

In terms of performance, both Telnet products for HP3000 incoming access (Telnet Express server & Telnet Access Card) have been used under heavy workload conditions. Two remarks are necessary :

a) It is very difficult to estimate the performance of either product. What is certain is that they behave differently. On a global scale, we are very satisfied (more specific performance figures appear later)

b) Due to our installation early in the product life cycle, we were not explicitly warned by HP about Telnet's performance impact in character-mode applications. It seems that HP's estimates of capacity are very conservative.

We have given HP the feedback that we see the need for more comprehensive tools for access control on HP's Telnet. Because our environment is large (Internet contains almost 3000 networks and about one million connected computers) it would make us feel more secure if we had more effective control mechanisms over Telnet users.

From a general standpoint, the HP3000 Telnet solution is flexible, allowing sites to start with one product and then add additional ones. This will be the case in some universities where the number of concurrent sessions will increase with time. This solution is also flexible in the sense that adding an HP3000 system will not require an additional Telnet product.

The management of a DTC network is simplified if they are all on the same software release. At release 10.5 , the DTC is no longer a terminal server dedicated to the HP3000, but rather also an Telnet/TCP/IP terminal server.

As of today, the main advice that we can give focuses on the performance of HP3000 incoming access. It is important to evaluate this aspect thoroughly before making final configuration choices. Your requirements need to be very carefully evaluated : number and usage characteristics of end-users, application behavior, and overall Telnet hosts to HP3000/900 traffic (in transactions/minute). This allows you to determine what you have to purchase, by comparing your Telnet traffic, the maximum response time you can tolerate.

Due to our application being character-mode one, performance could have been a problem. With VTLS application, the Telnet Express product can handle up to at least 70 simultaneous sessions without problems, when HP says that 12 sessions can be supported in the "worst case", (i.e. when applications exchange long and continuous streams of bits). Moreover, Telnet cards have been almost fully utilized (about 35-40 simultaneous sessions) with acceptable performance figures.

These high numbers are partly due to search behavior of library system users. After a search -with author, title, subject term etc..- they get an index listing on screen. It has to be read before either next screen of index or some references are chosen. Users are not sending or requesting data on a constant pace; they have to think about search strategies, value of references and so on.

4.7 General Feedback

HP's Telnet solution (Access to HP3000/900 from Telnet hosts and access from DTC to Telnet hosts) is now being used in more and more Universities sites. We are very pleased with the product, which meets our requirements.

In the area of access from DTC to Telnet hosts, we have found this functionality to be perfectly satisfactory, and we particularly enjoy the additional features of release 10.5 (new management functions, multi-session, etc..)

Access from Telnet hosts to HP3000/900 works also well. We would however, be interested in seeing some enhancements :

a) Increased access transparency for HP3000 applications accessing Telnet devices and systems , in particular the support of nailed devices and printers.

b) Security and resource control , which keeps track of access, connect time etc.. per session.

The first generation TCP/IP products are usually not noted for their performance figures. However, HP seems to have done well, which maybe is partly due to their earlier experience with TCP/IP in NS/3000. From the performance standpoint, the best results will probably come through hardware upgrades - which are fairly easy with the DTC. One might also with good reasons suspect that processing power on the DTC is cheaper than the processing power on the CPU. This is important since a NS/VT user -who needs TCP/IP conversions as well- needs roughly three times as much CPU and memory as the normal user.

In general, thee DTC is easy to use and easy to manage. The parameters that need to be configured are the ones which are important in a multivendor network. The total number of parameters to configure is limited, compared to other terminal servers which provide more than 50 configurable parameters per port. The DTC user interface is basic and sufficient. Our students don't need to learn complex & specialized commands: they want to access their application and run it, nothing else. If we compare HP's DTC to 3-COM CS terminal servers :

* DTC user i/f has 5 commands with only 2 parameters
* CS user i/f has 10 command with 55 parameters

From our perspective, the DTC provides these benefits :

(1) VTLS users don't have to have multiple terminals on their desks.

(2) A terminal connected to a DTC or to another system has transparent access to all LINNEA libraries and to other 250 library systems freely available in the Internet. Access is independent of the geographical location of the end-user and the application.

(3) The Library systems don't waste CPU power handling virtual terminal protocols (Telnet) as this is handled by the Telnet access card in the DTC. The HP3000 is completely dedicated to the application, and that was the objective when we sized the Library systems.

(4) The DTC coexists with other non-HP terminal servers. It is one alternative for terminal connectivity in FUNET and not only an HP3000 terminal server.

(5) We have the flexibility to add additional DTC services in the future based on new requirements. For instance some universities will add X.25 connectivity to existing DTCs. Others will increase the number of available sessions for HP3000/900 access. As of today, some libraries have already added X.25 connectivity to DTCs or plan to increase the number of existing Telnet products.

As a bottom line, this solution fits well in our strategy for end-user connectivity.

What are the areas of weaknesses of the DTC? The only one we see today is the lack of performance information in Telnet Access & Telnet Express products datasheet or manuals . The available information is sparse, which appears to defy explanation. The performance information should include performance rates for most common HP3000 applications, which could serve either as direct source of information or as general guideline for estimating performance. The worst case figures HP provides are not very helpful in the real life, as they seem to be overly pessimistic.

HP's Telnet solution has been somewhat difficult to explain to people who do not know HP3000 very well. Even if innovative, it brings the basic features. Again our recommendation to HP would be to provide more information about solution benefits (for example very easy management of Telnet and of the DTC, outstanding sharing capabilities).

5. CONCLUSION

Regarding the requirements listed at the onset of this document, HP's Datacommunication & Terminal Controller solution today provides universities with an excellent solution. All library systems are accessible from other systems on the same site, or from other sites and the DTC, like other Telnet/TCP/IP terminal servers, allows students to log on to non-HP systems. The following points result from HP's solution :

* Multivendor connectivity to 900 Series HP3000 systems and to non-HP systems
* Local and remote access for all end-users

It seems to us that HP is aggressively enhancing the DTC and HP OpenView DTC Manager products. New features and services are added to old ones on a regular basis. From the user's standpoint, it is beneficial to have a product which seems to have strategic importance for HP. Hence, confirming our selection of the DTC.

Even if some enhancements are needed, the HP solution is a good one and we can recommend it to other HP users .

PAPER #3874

DISK ARRAYS

Ed Pavlinik
Hewlett-Packard
Network Server Division
11413 Chinden Boulevard
Boise, Idaho 83707
(208) 323-2060

INTRODUCTION

Imagine having a totally new high capacity disk storage device attached to your computer system that is capable of **protecting your data** in the event of a disk failure! A disk drive may fail, yet the system still has uninterrupted access to all data and a user may even be unaware that a failure has occurred! Imagine being able to replace a disk drive while the system is still on-line. Think of the resultant increase in **data availability**!

Also imagine being able to **tune the performance** of your disk storage so that it can better meet the needs of your system. As your requirements change, the new disk storage system can be easily adapted to meet these changing needs.

High data availability, **tunable performance** and **increased disk connectivity** are just some of the benefits offered by a relatively new class of products called **Disk Storage Arrays.**

What makes the disk array different from traditional on-line disk storage devices? How can a data processing manager or workstation user take advantage of this new technology? This paper answers these questions, and provides an understanding of disk storage arrays as potential solutions to your changing mass storage requirements.

Disk arrays have recently become a leading topic of discussion in the computer systems business, not just among industry gurus, but also at trade shows and user group meetings worldwide. A variety of papers have been published and numerous articles have appeared in industry publications. The technology is now proven, and many users are already realizing the benefits of disk array technology with their HP systems.

WHAT ARE DISK ARRAYS?

The terms used to describe disk arrays can unfortunately lead to some confusion, since the phrase "**disk array**" has in the past been used to describe entirely different product concepts. For example, some products have been called disk arrays even though technically they might more properly be described as conventional disk storage systems or "just a bunch of disks". Despite possible confusion in terminology, the primary functions of a disk array are to **increase data availability, to increase total storage capacity**, and **to provide performance flexibility** by selectively spreading data over multiple disk mechanisms.

This paper will define a disk array as a storage subsystem consisting of multiple disk mechanisms under command of an array controller, incorporating several unique features that differentiate it from more traditional devices. These features are described in more detail in the following sections of this paper.

Data Protection

Most types of disk arrays provide an option for **protecting data** in the event of a disk failure. This becomes important as the number of disks on a system increases, thus increasing the likelihood of a failure. Normally, a disk crash brings the system down and removes it from service until the problem is located and repaired, and the data is reloaded. This can cause major problems since in many cases, the value of the data stored on the disk drives exceeds the value of the system.

The disk array is powerful enough to **immunize data from disk crashes**! In certain types of disk arrays, the array controller generates encoded data, which is comparable to parity or checksum calculations. This encoding scheme allows missing user data to be recreated via simple arithmetic, so that in spite of a disk failure, the data is still accessible. A major benefit is that the computer remains on-line and available for service with no downtime.

Other arrays offer **disk mirroring** as an option to protect data by maintaining two identical copies of all stored information. These types of arrays duplicate the data on redundant disks, so that a failure still allows access to the remaining disk drive and its data. The end result remains the same -- **high data availability!**

Increased Storage Capacity

The typical disk storage array is a mass storage system utilizing several small form factor disk drives, such as 5.25-inch or 3.5-inch, linked together with an intelligent controller to provide a significant amount of storage. Capacities of disk drives have been steadily increasing as manufacturers push the limits of magnetic recording technology. Grouping these drives together can provide substantial amounts of storage to meet the needs of most computer systems.

Since smaller diameter disk drives are being mass produced in large volumes, production economies of scale result in lower manufacturing costs. The disk array allows multiuser computer systems and workstations to take advantage of the benefits associated with smaller form factors, and offers many new and innovative capabilities not previously achievable .

Performance Flexibility

The disk array controller can be configured in a variety of ways to offer a high degree of flexibility in meeting diverse user requirements. One technique utilized by array controllers writes one or more bytes of information to each disk drive in a parallel fashion, allowing all to work in unison on a single data transfer. For performance reasons the disk drives are usually synchronized so that data is written to the same location at the same time, minimizing disk latencies. The result is an increase in the data transfer rate for long transfers, most noticeable for block sizes of 16K or larger.

Another array technique spreads out larger data blocks or multiple smaller data blocks independently to take advantage of overlapping disk accesses. Overlapping of disk accesses improves I/O performance for smaller data transfers.

Disk Array Performance

The primary benefit of disk arrays is to provide large amounts of storage at high levels of data availability, and in most cases, disk arrays will not provide performance gains over an equivalent number of non-arrayed disks. If all spindles are synchronized and the data is striped over all disks, data transfers take place in parallel. Therefore, the array has a speed advantage for large data transfers (16K or greater). Instead of one disk transferring data, the array has multiple disks transferring simultaneously. In this mode of operation the disk array appears to the host system as one large disk drive, and is addressed as one logical device.

In another controller design, the array may appear to the computer system as several unique disk drives all operating independently of each other. This allows for concurrent disk mechanism operation without data striping or parity, useful for very I/O intensive applications, since multiple I/O's can be executed at the same time. This concept is called Independent Mode of operation, and does not provide data protection, since there is no parity disk.

Different types of arrays can also achieve concurrent operation for certain operations by working together in pairs of a much larger group. Arrays of this type can process multiple small transfers simultaneously to speed up performance. In this example, the striping is done on a smaller number of disks in the group for small transfers and large transfers will keep all the drives busy in parallel. This will be explained in more detail in subsequent sections, but will be referred to as a Mixed Mode of operation.

As always, actual system performance depends upon a number of factors such as data block size, read/write ratios, random or sequential characteristics of the data, raw I/O or file system I/O, number of I/O's queued, and the total storage per system, which will influence the number of disk spindles. Memory management and caching will also influence whether a logical I/O will result in a physical disk I/O. If data blocks are cached in memory, the chances of a cache "hit" increase. When a cached hit occurs, the data is simply read from the cache without the need to perform another seek.

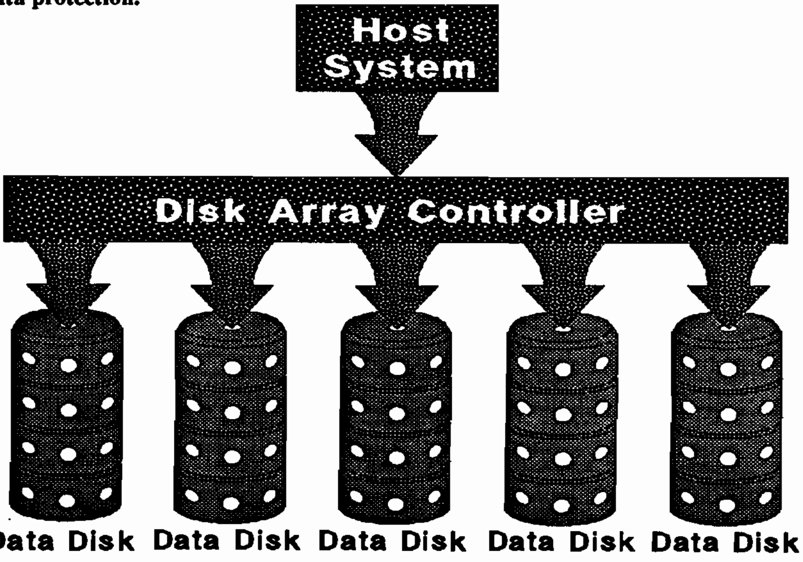
REDUNDANT ARRAYS OF INEXPENSIVE DISKS (RAID)

To understand disk arrays it is important to be familiar with the **RAID** concept. **RAID** is an new acronym associated with disk arrays first described in a University of California at Berkeley paper written by Patterson, Gibson, and Katz, entitled "A Case for Redundant Arrays of Inexpensive Disks, or **RAID**," published in 1987.

Many different disk array configurations are possible, depending on end-user requirements and the goals of the manufacturer. Each controller design has a different functionality to accomplish specific performance and data availability goals. Since only a few of these configurations are practical for most on-line transaction processing systems, servers, and workstations, RAID Levels 0, 1, 3, and 5 will be examined in more detail. We will also describe a method of array operation where each disk acts as an independent spindle.

INDEPENDENT MODE

Independent mode is designed to provide a high I/O rate disk array solution without data protection.



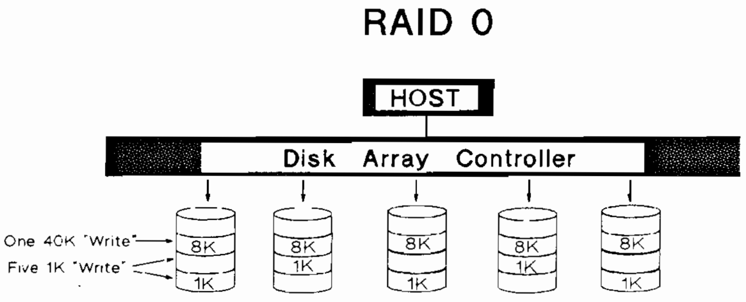
In this operating mode, the array of five disks appears to the host SPU as five separate and independent logical devices. Since each disk mechanism has its own address, each can process reads and writes independently of the others. The result is concurrency or overlap of I/O's which will produce a high number of I/O's per second. However, there is no data protection available with this operating mode, and there is no data striping, so that there is no multplicative effect on the transfer rates. The aggregate transfer rate of the disk array will simply be the transfer rate of a single disk mechanism. All the disk mechanisms share the same array controller.

A variant of this mode of operation is a mixed mode which incorporates a pair of 2-way striped disk mechanisms under command of the array controller. The array in this case appears to the host SPU as two logical devices.

RAID LEVEL ZERO

RAID Level Zero is designed to provide a high I/O rate disk array solution, without data protection.

The following diagram shows how a large forty-kilobyte "write" would be spread by the array controller among the disk drives. This large transfer requires that all five disk drives be available before it can execute. The drawing also illustrates five separate and independent one-kilobyte "writes" overlapping each other in time, serving to increase the I/O rate.



HEWLETT PACKARD

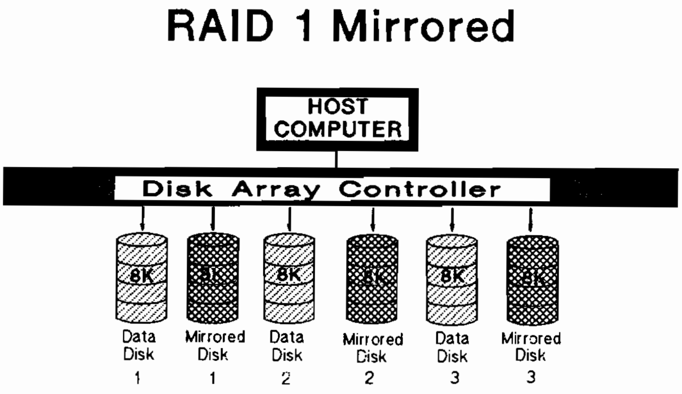
In this type of disk array the data transfer from the host system is split up by the array controller and subsequently spread across each disk drive into segments whose size depends on the defined block depth. Short data transfers may require only one or two disk drives, while longer transfers will utilize the entire group.

The advantage of a RAID Level Zero array is that several small transfers can be processed simultaneously to achieve a degree of overlap and resultant increase in I/O rate, since each group of disk drives can be working on a different I/O. In executing a workload of I/O's consisting of mixed length transfers from the host, all required disk drives must be available to execute longer transfers. A long transfer to an array that is working on several small transfers remains in the queue until all disk drives become available to process it. A disadvantage of RAID 0 is that **no data protection** is available with this type of disk array.

RAID LEVEL ONE

RAID Level One is designed to provide a high data availability disk array solution.

This type of disk array uses a configuration known as **disk mirroring,** in which a duplicate copy of the data in one disk drive is also stored on another disk drive. The following diagram shows a typical RAID Level One disk array:



Other implementations of disk mirroring duplicate not only the disk mechanism, but also the disk controller and power supply to provide a higher level of protection from the possibility of hardware failure. Some solutions will even mirror the entire array for an extra measure of data protection and even higher data availability. An obvious disadvantage of disk mirroring is the high cost of duplicating the disk drives, which makes the effective cost per megabyte twice that of a non-mirrored solution.

CONVENTIONAL NON-ARRAY MIRRORING

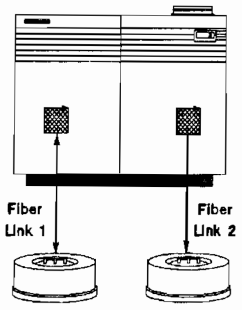
RAID Level One disk arrays differ from conventional disk mirroring solutions in that system level disk mirroring requires special applications software to manage the mirroring activities. Disk mirroring solutions will provide a higher level of availability than RAID Level One since everything is duplicated. The user purchases two complete sets of all disks, interface cards, cables, etc.

Here's an example of a system level disk mirroring solution, not using disk arrays. However, disk arrays can also be mirrored, for even higher levels of protection.

MIRRORED DISK OPERATION

Normal Operating Mode

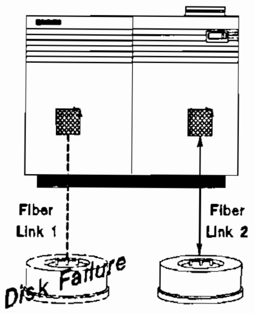
- Transparent To Users/Applications
- Minimal Overhead On Disk Writes
- Higher Performance On Disk Reads
- Simple Control & Operation



Data Duplicated On Mirrored Disks

When Disk Fails

- Transparent Switch On Failure
- Online Replacement Of Disk



Access to Second Disk

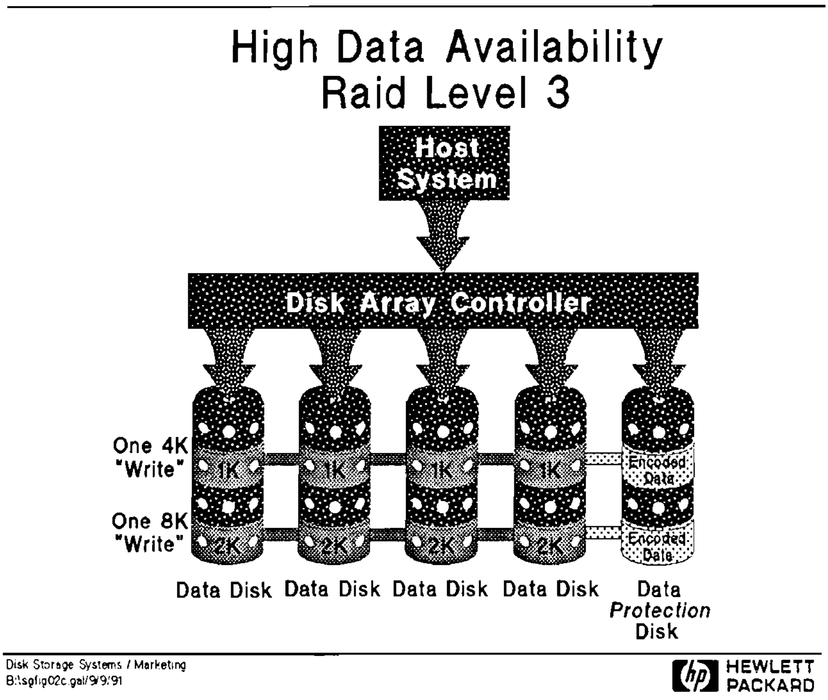
In the event of a disk failure, the special disk mirroring software will automatically switch all I/O activity for the mirrored pair of disks to the surviving disk in the pair. Repair and resynchronization of the failed disk drive can be done transparently to users and applications. The system knows that a failure has occurred via a console message, but the users and their applications will have the same access to the data as if nothing at all had happened. High data availability is a key user benefit from disk mirroring, since it extends system uptime by saving the system in the event of a disk failure. Mirroring has minimal overhead on disk writes, since two copies of the data and any changes have to be made. Higher performance can be achieved on disk reads however, since I/O can be processed by two disk spindles concurrently. This provides a performance benefit in addition to high data availability.

The disk mirroring solution duplicates the entire disk system, thus protecting the data against power supply, controller, fan, and cabling failures, in addition to a failure in the disk mechanism. The disadvantage of disk mirroring is the cost of duplicating the disk drives, making the effective cost per megabyte twice that of an non-mirrored system. Actually, the true cost may actually be more than twice as much due to the cost of the disk mirroring software which manages the entire operation. For data which must be protected at all costs and remain on-line in a high availability system, disk mirroring advantages will far outweigh the higher costs.

Disk mirroring also allows a system to be designed to achieve on-line backup. Since there are two disks storing every bit of data on the system, one disk in the mirrored pair can service on-line transaction processing, while the other can be dedicated for backing up to a secondary storage device. The only delay in the system is approximately five minutes of quiescent time at the beginning of the process to maintain data integrity. After the backup has been completed, the software schedules an on-line resynchronization of the mirrored disks.

RAID LEVEL THREE

RAID Level Three is designed to provide a high transfer rate, moderate I/O rate, high availability disk array solution.



This type of disk array uses a separate parity disk to store checksum data. The function of the parity disk is to store the EXCLUSIVE OR of the data kept on the data disks. This allows for a bit-by-bit comparison and subsequent reconstruction of the data in the event of a failure of one of the data disks. The disk spindles are synchronized so that at a given point in time all the heads in the group of disks are reading or writing on the same sector location in parallel, thus saving disk latencies and improving performance.

Data is spread or striped across all the data disks in the array on a byte-by-byte basis, and the array appears to the system as a single large disk drive with a single logical unit address. During normal operation the array transfers data in a parallel fashion at the theoretical rate of a single mechanism multiplied by the number of data drives. Actual transfer rate depends on the host bus adapter bandwidth and system data patterns and workload. For some applications, a performance improvement may be achieved due to the higher transfer rate, but this will be most noticeable with data blocks of 16K or greater. Every write operation will involve all disks in the array, since new parity will be written to the parity disk. Reads involve all the data disks in the array.

Buffer caching is often used with this type of disk array to provide performance benefits for sequential "reads" that can be cached and may provide instant access for data that is sequential in nature. This may have a dramatic effect on increasing the number of I/O's per second with a RAID 3 disk array, as long as the data is localized in a cylinder group or sequential in nature. The caching can in certain sequential "read" applications nearly triple the I/O rate!

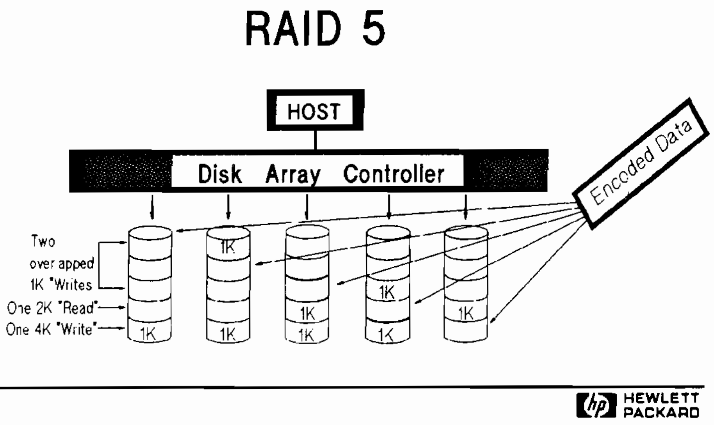
In the event of a disk failure, the array controller reconstructs data that is missing from the failed disk through the use of the parity information. There will be no loss of performance as the controller is designed to accomplish the extra tasks involved. The array can operate in this mode with no loss of efficiency until the chance occurrence of another disk failure, but statistically the probability of that happening is very small. The system knows that a disk failure has occurred, and the failed mechanism can be replaced at the next service call or else during a slack period in the system operation. A new drive can be inserted to replace the faulty unit and the array controller will rebuild the data on the new disk on command. During all this activity, the system is still available for on-line transaction processing or batch applications.

Some disk arrays facilitate the "hot" replacement of the disk mechanism and are robust enough to permit many thousands of insertion/withdrawal cycles with no damage to any of the hardware or connector pins. This represents a tremendous contribution to system uptime, since power does not need to be turned off to replace a disk array mechanism. Other lower availability designs require a power off cycle and sheet metal removal to get at the disk mechanisms, and then cables need to be manually disconnected..

The cost per usable megabyte of storage in this type of disk array increases due to the dedicated parity disk which cannot be used for storage of user data. For example, in a four-way striped parity array, there is 25% overhead in storage costs. This is due to the fact that the product consists of five disk mechanisms, but user data can be stored on only four.

RAID LEVEL FIVE - DATA EMBEDDED WITH PARITY ACROSS ALL DISKS

RAID Level Five is designed to provide a moderate transfer rate, moderate I/O rate, high availability disk array.



This design is more complex than the RAID Level Three disk array. Raid Five disk arrays exhibit a performance penalty on disk "writes", due to the method used to calculate and write parity information. This extra overhead associated with "writes" may degrade RAID 5 performance, depending on the read/write ratio demanded by an application.

The data is "block" striped across all the disks in the group, but the parity information for each sector is not stored on the same disk. The array controller manages the generation and location of the parity information for each sector stored. For example, the controller parity and data storage sequence might be as follows: for sector zero the parity data is stored on disk one, sector one's parity is stored on disk two, sector two's parity is on disk three, sector three's on disk four, etc.

This Level Five array will have decreased "write" performance, since for a write involving just a few disks in the group for a small block of data, all the disks in the group need to be read, new parity calculated, and then new parity information rewritten. This "READ-MODIFY-WRITE" cycle represents extra overhead for writes, when compared to a level three design. On "reads" involving just a few disks, read concurrence occurs, since the array may be processing multiple I/O's to different disks in the group. In the event of a disk failure in a RAID Level Five array, the missing data is calculated from the parity or checksum information in the same fashion as in RAID Three.

WHAT ARE THE BENEFITS OF DISK ARRAYS?

High Data Availability

Since increasing numbers of disk drives are being connected to computer systems, servers, and workstations due to steadily growing storage requirements, keeping these systems operational becomes more important. This is particularly true as more and more systems require operation twenty-four hours per day and seven days per week. Disk drive failures and resultant system halts often cannot be tolerated. Critical applications cannot afford system downtime.

As explained earlier, many disk storage arrays offer **data protection** which guards the data against potential loss due to a drive failure. This is accomplished by dedicating an entire disk drive or sections (depending on the implementation) of all disk drives to storage of encoded information. This allows the array controller to recreate data from a faulty drive with no loss in performance. Other arrays duplicate or mirror the disk drives to provide two copies of the data in case one copy is lost.

Certain types of arrays also offer the capability to replace faulty disk drives in a "**hot plug**" fashion with no loss of system availability. The new drive is simply inserted and the array controller "rebuilds" it with the original data. This activity is managed entirely by the disk array controller. As a result, the computer system can tolerate disk drive failures and their subsequent replacement with no loss in overall system availability. Freedom from the dreaded fear of data loss is a key benefit of disk arrays.

Increased Disk Storage per System

In certain RAID levels, the disk array appears to the host system as a single large, logical device. For systems which are constrained on the total number of logical devices, the array offers a vehicle to increase total disk storage per system. For example, a non-arrayed disk storage system may have a maximum capacity of 1 Gbyte per logical device. A disk array using the same 1 Gbyte disk drive may be configured to provide 4 Gbytes per logical device. The end result is a fourfold increase in the amount of disk storage per system through use of arrays.

Large Capacity Storage in a Small Footprint

Disk arrays offer large amounts of storage in relatively small cabinets, resulting in more storage per unit of floorspace. With high density, small form factor disk drives sharing the same controller and power supply, an array offers substantial floor space savings over more conventional storage alternatives.

Increased Flexibility through Intelligent Array Controllers

Some disk arrays are quite flexible to meet changing storage requirements. They can be configured for high data transfer rates, data protection, or optimized I/O. In fact, some designs allow switching among various modes of operation. This allows the user to store data with different access characteristics on a different type of array. For example, data that doesn't often change may not require data protection. Databases with large size transfers may better utilize arrays with high data transfer rates. I/O-intensive data may be best suited for transaction oriented arrays. These data segmentations are a few of the possibilities available to array users. Operating modes, however, can be modified as requirements change, although some mode changes will require a system reload to reformat the data in the correct configuration.

Performance Enhancements

Since disk arrays differ with regard to operating characteristics, certain performance implications emerge as a result. Many arrays are designed for optimal system performance with specific computer architectures. For example, the physical layout of data on disk drives in the array can cause the transfer rate to be increased significantly. Other designs optimize the number of transactions per second through concurrent accesses. Other array controllers utilize buffer caches to speed up performance by eliminating the need for a disk seek for sequential data arranged in the same disk cylinder. Contiguous data is simply routed into the buffer cache, where a "hit" will provide instantaneous access. This feature can potentially triple the I/O rate in certain applications, depending on the size of the buffer and the disk transfer blocks.

How well these enhancements are leveraged into overall system performance improvements depends on how well the disk array has been integrated into the system, and is a function not only of the system characteristics, but also the end-user application.

Although a major function of the disk array is to provide a large amount of storage at a high level of data availability, performance gains may also be realized for certain system applications. Through use of an intelligent controller managing the operation of several disk drives, it is possible to achieve performance gains in a number of different ways. If all disk spindles are synchronized and data is distributed over all drives, data transfers occur in a parallel fashion. Therefore, the array has a potential speed advantage for large data transfers. Instead of one drive transferring data, the array has multiple drives transferring data simultaneously. The major benefit of this arrangement is a high transfer rate of data to and from the disk array. In this mode of operation, the disk array appears to the host system as one large disk drive.

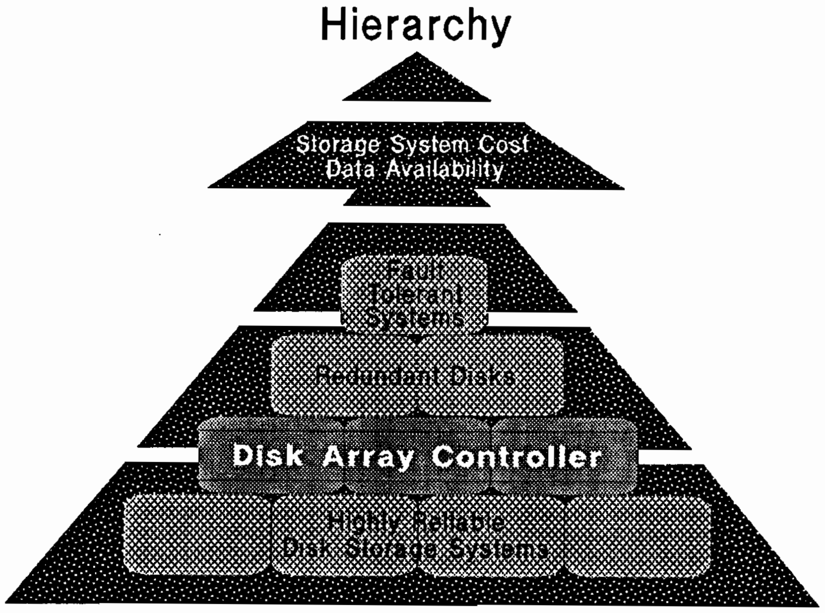
In yet another type of controller design, the array may appear to the computer system as several standalone drives all operating independently. This allows for concurrent disk accesses which is beneficial for I/O intensive applications. The key user benefit in this configuration is the ability to service a large number of I/O's per second.

Some types of arrays can achieve concurrent I/O activity by working in pairs within the array. For example, an array might consist of four disk drives, each pair representing a single logical device to the host computer. Arrays of this type can process multiple small transfers simultaneously to speed up performance. In this example, the file spreading is done across two drives only. The advantage offered by this design is a balanced combination of high transfer rate and concurrent I/O.

DISK ARRAY POSITIONING

How does the data availability of disk arrays compare with other solutions? A good way to visualize the concept of disk array data availability is to consider a **hierarchy of data protection**. In this example, higher data availability is achievable via various techniques for a corresponding increase in cost. For each desired level of availability, a specific storage solution can be selected.

The following example shows various levels of availability, starting from highly reliable disk storage systems and working upward to progressively greater data availability. As you might expect, costs will increase as you move toward the top of the pyramid. Many applications, however, demand these high availability systems since the benefits outweigh the costs. The deciding factor for selecting a high availability solution is really the **cost of system downtime** which is unique for every business.



Disk Storage Systems / Marketing
B:\sgfig06c.gal/9/9/9

HEWLETT PACKARD

Disk Drive Reliability may be considered a cornerstone or foundation in this hierarchy upon which additional layers of data protection may be constructed. Continual improvements in disk drive reliability have made possible the design of disk mechanisms with a Mean Time Between Failure Rate (MTBF) in excess of 150,000 hours. However, large MTBF does not guarantee failure-free operation, but only implies that failures will occur less frequently on the average.

Near the top of this hierarchy are **redundant disks** which can be used to provide insurance against disk drive failure. Some systems duplicate the entire disk storage system to maintain two copies of each set of data. In this solution, fans, power supplies, controllers, cables, and host bus adapters are duplicated, removing many single points of failure. Special software is required in this case to switch the applications to the desired set of drives and to keep the data synchronized.

Additional fault tolerant features can be added to redundant disk storage to obtain further improvements, making this type of storage most suitable for fault tolerant systems that feature almost no downtime. Some systems may even feature triple redundancy for extremely critical applications. Power sources can be duplicated and some systems even have diesel generators to provide emergency power independent of the local power grid. These systems represent the ultimate in high data availability for use in mission-critical applications with continuous processing, however, these systems are also extremely costly.

The intermediate level in this hierarchy consists of **disk storage arrays with data protection**. This middle section offers solid levels of data availability for less cost than the fully redundant solution. Disk arrays do not duplicate every component like redundant systems, but their basic goal is to provide protection for higher failure rate components. At a minimum, the disk drives themselves are protected with mirroring or else via the RAID Three or RAID Five encoded data. Some arrays also offer dual controllers, power supplies, and fans. Many applications can take advantage of the level of data availability offered by disk arrays and the lower cost is an attractive feature when compared to full redundancy.

SUMMARY

Disk arrays represent a major step forward in the quest for **higher data availability** and are designed to meet user requirements for less cost than fully redundant or mirrored disk storage systems. User demands for increasing quantities of disk space often conflict with the need to achieve higher system availability. The disk array allows the user to access more storage with higher availability and higher throughput. Various types of arrays can be configured to provide both high transfer rates and concurrent disk accesses.

The emerging disk array products are flexible mass storage devices that meet a wide variety of computer user requirements. Multiuser system and workstation users alike can benefit from **data protection, high data availability**, and **tunable performance** offered by disk arrays. Conceptual disk array subsystems as described by the various **RAID** levels will easily support future enhancements designed to increase flexibility and versatility for the end user. As a result, the future looks quite appealing for a wide variety of disk array products, each improving on certain aspects of performance, or data availability, or both.

Hewlett-Packard has recently introduced several products utilizing disk array technology for multiuser computer systems, workstations, and servers, incorporating many of the features described in this paper. These Hewlett-Packard disk arrays are shipping in production volumes at this time.

GLOSSARY OF DISK ARRAY TERMINOLOGY

BLOCK DEPTH

A specified number of sectors allocated to each disk drive in an array for data spreading in RAID Levels Zero and Five. This establishes the depth with which data is written to each disk drive in an array and can be modified to adjust performance. It will affect the degree of concurrency of operation.

BUFFER CACHE

Temporary storage space either on the disk array controller or on the disks themselves which can be used to store multiple contiguous blocks of data to provide dramatic performance improvements for localized data.

CONCURRENCY

The simultaneous execution of I/O's among a group of disk drives. This allows for increased I/O rates due to multiple actuators, in that several drives may be seeking or transferring data independent of each other.

DATA AVAILABILITY

A measure of the amount of time that data remains accessible to users of a computer system. In a data protection disk array, data may remain accessible even though a disk drive has failed.

DATA PROTECTION

A technique that allows data to remain accessible despite the occurrence of a disk drive failure. Some vehicles used for data protection include disk mirroring as well as specially encoded data that allows for the recreation of missing data.

DISK ARRAY CONTROLLER

The hardware and firmware that regulates the flow of data to and from the host computer to the disk drives in a disk array.

DISK MECHANISM

The basic building block of a disk storage subsystem. The disk mechanism includes everything required to store data except power supply, fan, host bus adapter, and controller, although most mechanisms come with a SCSI or ESDI controller card packaged within the form factor.

DISK MIRRORING

A technique which duplicates disk drives to guard against a failure. This can be implemented in a disk array or else via system applications software.

FAULT TOLERANCE

This design feature is often built into mission critical systems which duplicate or even triplicate critical components to permit operation in the event of a failure. The result is a very high availability system that can tolerate many failures with no loss of functionality.

I/O'S PER SECOND

A performance measurement that indicates the average number of "reads" or "writes" that can be processed by the disk storage system per second. It is a function of the disk drive seek time, controller overhead, disk latency, transfer block size, and time required to transfer the data.

LATENCY

A measure of the rotational time required by the disk drive for positioning a specific sector on a disk platter under a read/write head. The average latency is defined to be the time required for half a rotation of the disk platter, usually measured in milliseconds.

OVERLAPPING READS OR WRITES

Simultaneous execution of reads or writes among disk drives in an array, resulting in increased I/O rates since the drives are operating independently of each other. Used to characterize RAID Level Zero or RAID Level Five with short data transfers.

QUEUE

A line of I/O transactions waiting to be processed.

RAID

An acronym for Redundant Arrays of Inexpensive Disks, first described in a paper published at Berkeley. Many variants of the architecture exist, some offer data protection in the event of disk drive failure.

SEEK TIME

The amount of time required for a read/write head on a disk drive to position itself on the correct track on the disk media, usually measured in milliseconds. An average seek is usually defined the time needed for a disk actuator arm to traverse one-third the full stroke distance.

SPINDLE

Another term for disk drive, derived from the part of the disk mechanism around which the platters spin.

SYNCHRONIZED SPINDLES

A technique employed in some disk arrays that insures that all index marks of the disk mechanisms are in alignment at the same point in time. This feature eliminates extra latencies.

TUNABLE PERFORMANCE

A feature offered by some disk arrays which allows the user to select the operating mode or RAID level. This allows a disk array to be tailored to better meet its data storage requirements.

TRANSFER RATE

A measure of the speed with which a block of data moves between a disk subsystem and main memory, usually measured in megabytes or kilobytes per second.