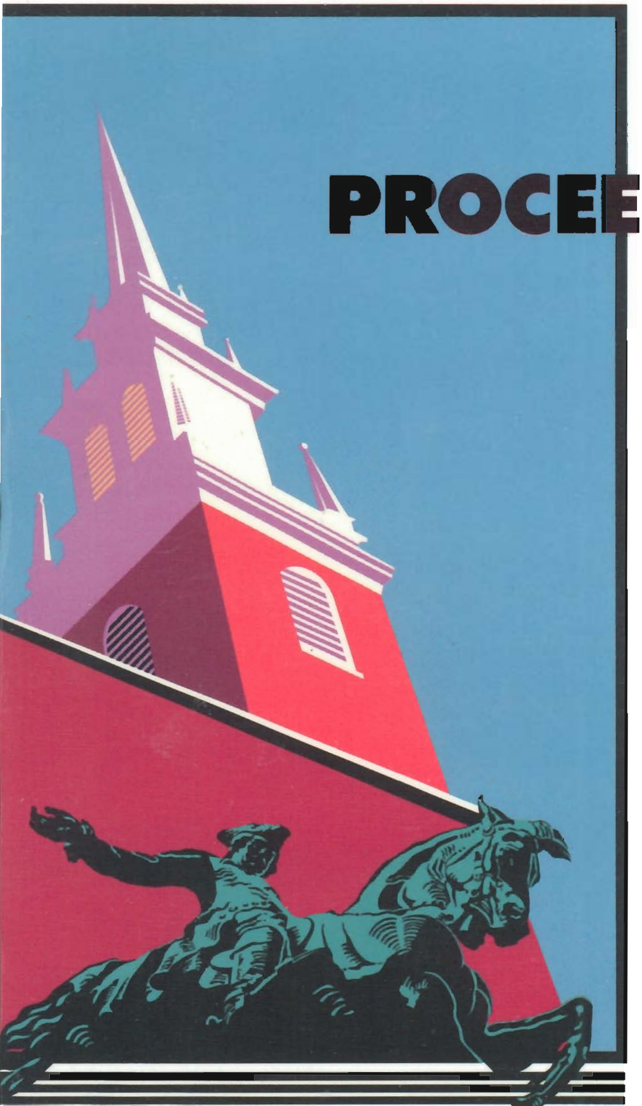
PROCEEDINGS



INTEREX
HP Users
CONFERENCE
AUGUST 20-23

RTE, HP-UX,
Workstations

BOSTON • 1990

NTEREX HP USERS CONFERENCE

AUGUST 20-23

Sponsored by **INTEREX** The International Association of Hewlett Packard Computer Users

INTEREX

The International Association of Hewlett-Packard Computer Users

PROCEEDINGS

of the

1990 INTEREX HP Users Conference

RTE, HP-UX, Workstations

at

Boston, Massachusetts
August 20-23. 1990

F. Stephen Gauss, Editor

Index by Paper Number

Index by Paper Number

Index by Author

UNIX

RTE SYSTEMS

TUTORIALS/ROUNDTABLES

ACCESSING MULTIPLE SHAREABLE EMAS IN A SINGLE PROGRAM

Warren R. Weber
Sandra S. Millford
AGS Genasys Corporation
9710 Patuxent Woods Drive
Columbia, MD 21046
(301) 596-7410

ABSTRACT

In a previous paper (Interex 88, Paper 26), the telecommunications Automatic Monitoring System (AMS) was described. AMS provides the ability to remotely monitor telecommunications equipment and detect equipment faults. As the size of the communications centers increased, the demand on the system exceeded the capabilities of a single computer. Distribution of the system required LAN interface software and an increased amount of shared data. When Shared Extended Memory Area (SHEMA) requirements for the new system were considered, they were found to exceed the system limitation of 1,022 pages; therefore, programs needed to access multiple SHEMAs.

This paper describes the design issues considered during the implementation of the new system. It includes a discussion of the program/operating system interface and the code required to allow a program to access more than one SHEMA.

1. Distribution of the Existing System

One must consider the type of system being implemented in order to understand why multiple Shareable Extended Memory Areas (SHEMAs) must be accessed by one program. As discussed in a previous INTEREX paper (INTEREX 88, Paper 26), the telecommunications Automatic Monitoring System (AMS) is a real-time, on-line, data collection system supporting telecommunications operations and maintenance functions. Designed to improve the quality and timeliness of information regarding the status of communications circuits, AMS provided operations staff the capability to quickly initiate the proper corrective actions to restore circuits to normal operations. By continuously monitoring the condition of all its circuits, AMS is capable of detecting, analyzing and reporting the presence of an anomalous circuit event on a real-time basis. It also provides information on the occurrence and duration of anomalous events to the technical support staff.

The current AMS system supports various combinations of HPIB and serial interfaces (generally either three HPIB and five mux cards or four HPIB and four mux cards). As is the case with most successful systems, AMS is a victim of its own achievements. With more and more monitoring requirements being levied against it, it is evident that the requirements will outstrip the resources available to a single A900. Currently, we are running out of system memory, backplane slots and processing power.

To alleviate this problem we are implementing a distributed architecture with a controlling A900 and equipment monitoring A400s. The A400s will provide the data collection and filtering functions of the system while the A900 will provide the operator interface and database functions. To allow for easy future additions to the configuration, we elected to use NS/1000 to support communications between the nodes.

2. Shared Data Requirements Increased

Although the distributed AMS (DAMS) will provide for greater processing capabilities, it also increases the amount of data to be stored. The new data pertains to the network configuration and interface, and additional alarm points associated with the the new monitoring equipment. Additionally, this data has to be shared between many programs. After completing the initial design for DAMS, it was discovered that, with a fully configured system, a control node's shared data requirements exceeded the RTE-A system limitation of 1,022 pages of SHEMA.

One solution considered was to use class queue I/O to share data. This method would forward data from one program linked to a SHEMA to another program linked to another SHEMA via a class queue. There were three major problems with this approach: class queue I/O was not resource efficient for our system given the amount of data involved; coordinating data transfers between programs proved to be impossible in a real-time environment; and some programs required access to data in both SHEMAs. Therefore, an algorithm was implemented to allow a program access to a SHEMA other than the one which it was linked.

3. Programs Allowed Access To Multiple SHEMAs

When a program is linked to a SHEMA, its ID segment contains two vital pieces of data referencing the SHEMA (Figure 1). First, the size of the SHEMA with which the program is linked resides in the $EMAS field (34th word) of the program's ID segment. Second, the SHEMA table entry number resides in the $MSEG field (35th word) of the ID segment. The SHEMA table (Figure 2) is a system table identifying all SHEMA partitions in the system. The number of entries in the table is predetermined, and the table is allocated at system generation. As each SHEMA is allocated, it is associated with an entry number specifying an offset from the beginning of the table.

By manipulating the $EMAS and $MSEG fields, a program may access data in a SHEMA other than the one with which it was originally linked. To perform this task, all that is required are the address of the program's ID segment and the SHEMA table entry number.

To perform this change, we have written an algorithm, Change EMA (ChgEMA), that modifies the $EMAS and $MSEG fields of the calling program's ID segment and causes the working map registers to be reloaded.

3.1 $EMAS

The last ten bits in the $EMAS field specify the size of the EMA being used. When a program is executed, this value is compared to the number of pages in the block of memory allocated to the program. If the value in $EMAS is larger than the value in the program's Memory Descriptor (MD), an error EM90 is generated; if the value in $EMAS is the same size or smaller, however, the program will be run.

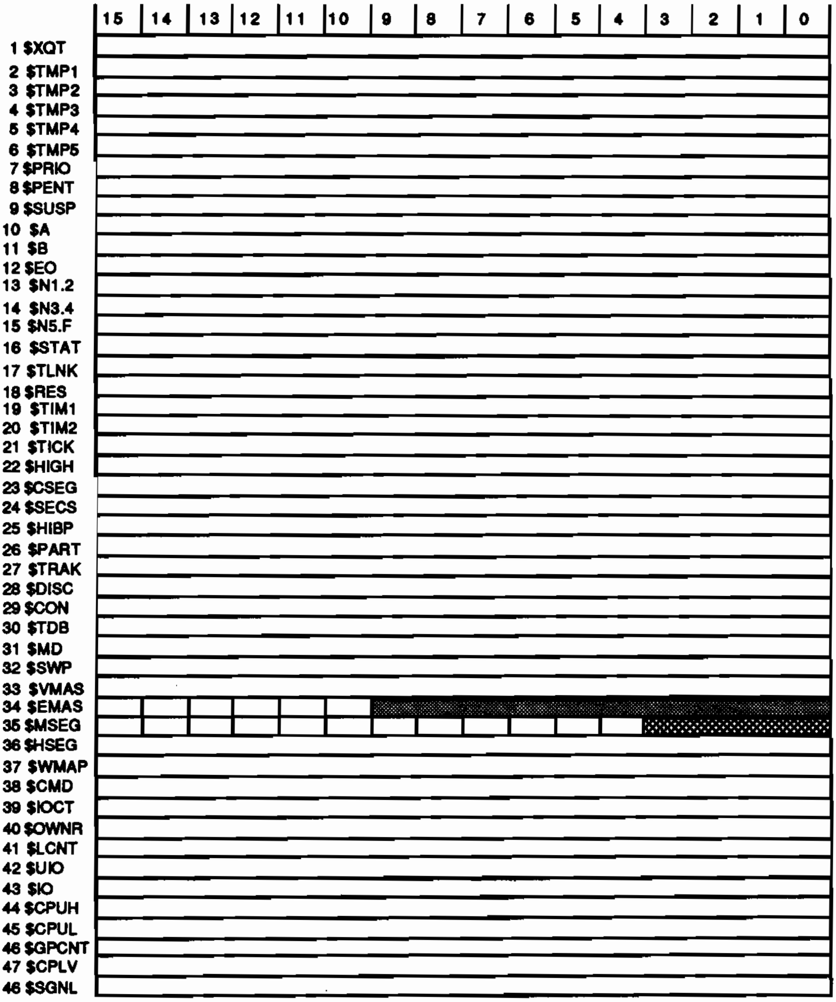


FIGURE 1 - ID SEGMENT

<table>
<tr><th></th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>Word 0</td><td colspan="16">Address of Memory Descriptor describing allocated space</td></tr>
<tr><td>Word 1</td><td colspan="16">SHEMA label, characters 1 and 2</td></tr>
<tr><td>Word 2</td><td colspan="16">SHEMA label, characters 3 and 4</td></tr>
<tr><td>Word 3</td><td colspan="16">SHEMA label, characters 4 and 6</td></tr>
<tr><td>Word 4</td><td>LK</td><td colspan="3">Reserved</td><td colspan="6">Number in system count</td><td colspan="6">Use count</td></tr>
</table>

FIGURE 2 - SHAREABLE EMA TABLE ENTRY

<table>
<tr><th></th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>Word 0</td><td colspan="16">Number of pages in block</td></tr>
<tr><td>Word 1</td><td colspan="16">Starting page of block</td></tr>
<tr><td>Word 2</td><td colspan="16">0 => In use as SHEMA partition</td></tr>
<tr><td>Word 3</td><td>C/D</td><td colspan="12">Reserved</td><td>IO</td><td>LK</td><td>OV</td></tr>
<tr><td>Word 4</td><td colspan="16">Pointer to next free block</td></tr>
<tr><td>Word 5</td><td colspan="16">Pointer to previous adjacent block</td></tr>
<tr><td>Word 6</td><td colspan="16">Pointer to next adjacent block</td></tr>
</table>

FIGURE 3 - MEMORY DESCRIPTOR

Using the table entry number supplied by the calling program, the address of the desired SHEMA's entry is obtained by adding the beginning address of the table to the table entry number minus one multiplied by the number of words in a SHEMA entry to the beginning address of the SHEMA table:

$SHTB + ((new table number - 1) * $SHSZ)

Using this address, the address of its MD is obtained from its first field. The number of pages allocated for the SHEMA is then read from the MD (Figure 3). The entire $EMAS value is read from the ID segment and the lower ten bits are cleared. The value obtained from the first word of the MD is OR'ed in, and the value is stored for later use.

3.2 $MSEG

The last four bits in the $MSEG field specify the table entry number of the SHEMA being used. This number is used by the dispatcher to locate the SHEMA table entry for the requested SHEMA.

The value of $MSEG is read from the program's ID segment and the lower four bits are cleared. The new table entry number is then OR'ed in and the value is stored for later use.

3.3 Level 1 Code

While reading information from a program's ID segment is not a privileged operation, writing information to an ID segment is a privileged operation requiring Level 1 code.

Level 1 code operates in privileged mode and is entered via a call to the $LIBR subroutine. When executing Level 1 code, the RTE-A Memory Protect facility is disabled as well as all normal I/O interrupts (only privileged interrupts will be serviced while executing this code). For this reason, it is crucial to minimize the time spent executing Level 1 code. As suggested by HP, any Level 1 code should be limited to 1 millisecond. The ChgEMA algorithm consists of four instructions executed in a Level 1 environment, taking approximately 10 microseconds.

ChgEMA does as much processing as possible prior to executing in a Level 1 environment. In fact, the only processing left to be done is to write the modified $MSEG and $EMAS fields back to the calling program's ID segment. To accomplish this, two cross-store commands are used. The new EMA size is written into the $EMAS field and the new SHEMA table entry number is stored into the $MSEG field. The $LIBX subroutine is then called to restore ChgEMA to unprivileged mode. A summary of the processing is shown in Figure 4.

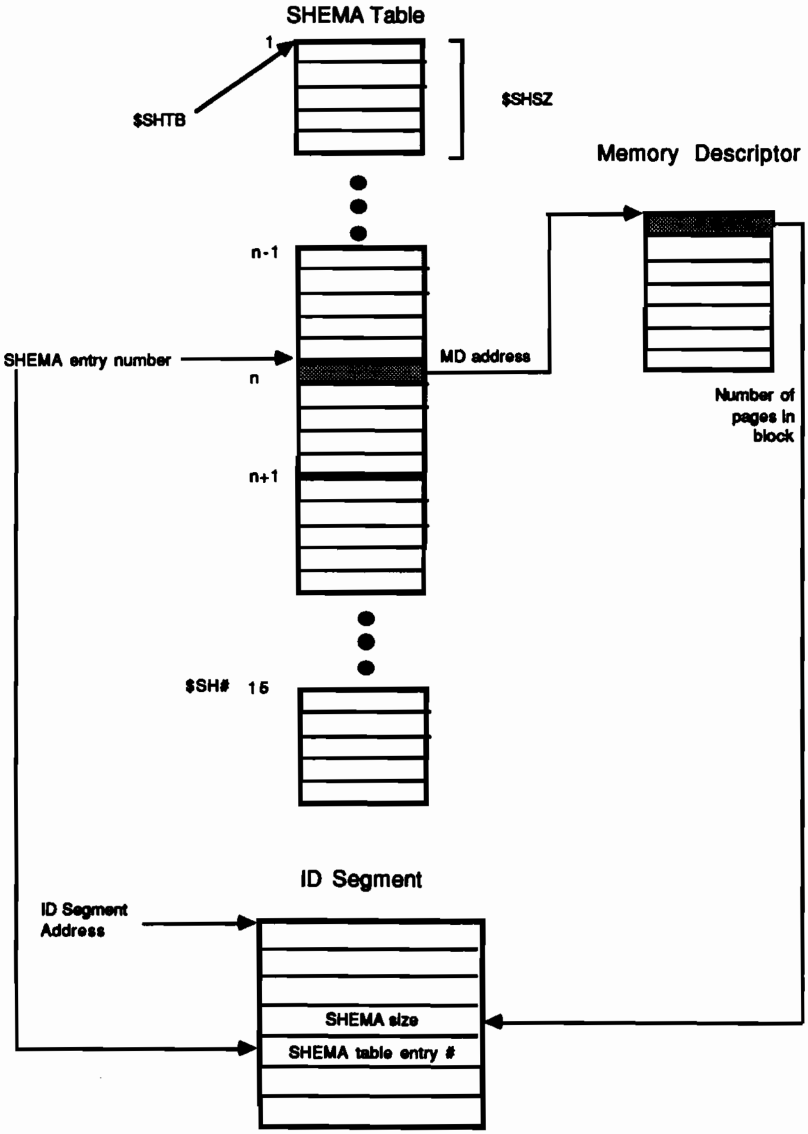


FIGURE 4 - PROCESSING OVERVIEW

3.4 Context Switch

Once the changes have been made to the ID segment, the dispatcher must reread it to update the working map registers. To do this, a context switch must be generated. This can be accomplished a number of ways including I/O, scheduling a program with wait, and sleeping. Each of these methods was investigated and putting the calling program to sleep for one clock tick was deemed to be the fastest. Sleeping causes the calling program to be taken off the run list and put on the time list; when the timer expires, the dispatcher reads the ID segment, sets up the working map registers using the new SHEMA and restarts the program.

3.5 Program Interface

The only stumbling block to implementing changing EMAs in an operational program is knowing the table entry number of the SHEMA you wish to access. We have written another program (GETSHEMA) that accepts a SHEMA name, scans the SHEMA table for a match (Words 1 through 3, Figure 2) and returns the entry number. This is performed for all SHEMAs to be accessed and the table entry numbers are stored in a data structure in each SHEMA for all programs to access (including those programs that have changed to new EMAs).

3.6 Implementation Consideration

Since ChgEMA modifies the program's ID segment, the user must insure that the system keeps its internal tables in order. When a program using a SHEMA is dispatched, the Use and In System counts are incremented in the appropriate entry in the SHEMA table. When a program terminates, the operating system checks the ID segment for SHEMA references and decrements the counts. If a program begins operating in SHEMA #1, changes to SHEMA #2 and then terminates, the SHEMA table will show (assuming there are no other programs using the two SHEMAs in question) a Use and In System count of 1 for SHEMA #1 and a Use and In System count of 63 (both counts are represented using six bits, therefore subtracting one from zero would set all six bits making 63) for SHEMA #2. Should another program linked to SHEMA #2 attempt to start up, it will abort since there are already 63 programs using SHEMA #2.

This problem is avoided in the DAMS code by making sure that all programs that call ChgEMA change back to the SHEMA they were linked to before they terminate.

4. Performance of the ChgEMA Algorithm

After developing the algorithm for accessing multiple SHEMAs, benchmark routines were written to determine its efficiency. The algorithm needed to be fast enough to appear transparent to the established code, thus MACRO code was implemented. Since a context switch is required to force the operating system to reload the modified ID segment, the algorithm was first coded as a program. However, restoring and scheduling a program takes a considerable amount of time, so the algorithm was then coded as a subroutine. The subroutine was discovered to be about 16 times faster than the program.

Processing times indicated below are for one "round trip", where a round trip is defined as changing from a program's original SHEMA to another and back again. The following benchmarks were performed on HP1000 A400 computers several times, with a varied number of trials each time.

4.1 Algorithm as a Program

The non-CDS MACRO program is invoked by a CDS Pascal program using the restore and schedule process. The MACRO program is restored with the FMPRPProgram command and is scheduled with an Exec 9 (schedule with wait) command. Two parameters are passed to the program: the ID segment address of the calling program and the SHEMA table entry number of the SHEMA which the calling program needs to access. The ID segment address can be obtained via the library function MyIDAdd which returns the one word address of the ID segment of the calling program. The SHEMA table entry numbers available for use are located in all SHEMAs used by AMS (having been placed there by GETSHEMA when the system is brought up). The parameters are passed as scheduling parameters.

The MACRO code retrieves the scheduling parameters with the RMPAR call. The specified SHEMA table entry number and size are placed into the ID segment of the invoking Pascal program. Since the MACRO program was scheduled with wait, a context switch occurs when returning control to the Pascal program. This causes the operating system to load the working map registers with the data for the new SHEMA, including the starting address of the SHEMA to be accessed. The body of the MACRO code is equivalent to that of the subroutine.

One round trip with the program implementation requires approximately 330 milliseconds.

4.2 Algorithm as a Subroutine

The non-CDS MACRO subroutine is called by a CDS Pascal program. Two parameters are passed to the subroutine: the ID segment address of the calling program and the SHEMA table entry number of the SHEMA which the calling routine needs to access. The calling program obtained these parameters as described in Section 4.1.

The MACRO subroutine code modifies the program's ID segment just as the program does; however, the subroutine must force a context switch by invoking the operating system via an executive command. The Exec 12 (sleep) command is executed, causing the Pascal program to sleep for one tick of the system clock. After the specified time has elapsed, the operating system reloads the working map registers with the data for the new SHEMA, including the starting address of the SHEMA to be accessed.

One round trip with the subroutine implementation requires approximately 20 milliseconds.

5. Summary

With more and more requirements being levied against the existing AMS system, the distribution of AMS was inevitable. The distributed system brings greater processing capabilities, and, unfortunately, an increased amount of data to be stored. This new data was to be shared by several programs, yet the data for a fully configured system exceeded the SHEMA system limitations. Therefore, the shared data was divided into two SHEMAS - one containing alarm point and associated monitoring equipment data, and one containing network configuration and interface data. To further complicate this matter, some programs required access to both SHEMAs.

The ChgEMA subroutine was written to permit access to more than one SHEMA by a single program. SHEMA access is obtained through the program's ID segment (which contains the size and the table entry number of the SHEMA in the 34th and 35th words, respectively). By modifying these fields to reference another SHEMA, it is possible to access another memory area. Modification of an ID segment requires privileged operation, as well as the reloading of the working map registers. A context switch causes the ID segment data to be reexamined and the working map registers to be reloaded. Thus, the ChgEMA subroutine permits programs to exceed the SHEMA system limitation and access other memory areas with which they are not linked in a timely, efficient manner.

6. References

1. Arbogast, Charles R., Telecommunications Automatic Monitoring System, Technical Proceedings, Interex 88, August 1988.

2. Hewlett-Packard, RTE-A System Design Manual, fourth edition, January 1989.

Lilly Fermentation Process Control System
Input/Output Driver Development

Lincoln Brill
Fermentation Development Automation
Lilly Research Laboratories,
Division of Eli Lilly and Company

The Lilly Fermentation Process Control System (LFPCS) was recently upgraded to an HP A900 RTE-A computer system from an HP E-series system. This project included developing new input/output (I/O) drivers for the existing front-end control equipment (Foxboro, Moore, Opto22 and others). The driver development consisted of four key phases: planning, designing, testing and reevaluating. The planning stage consisted of choosing the appropriate software language(s), completing thorough research on the front-end equipment and setting goals. The designing phase required the driver software to be logically organized and then produced. During the testing phase, which coincided with the design stage, the timing sequences were verified for proper communication and the software for valid I/O functionality. The reevaluation stage overlapped all the other stages, providing a higher quality driver on completion. Both parallel and serial I/O drivers for the LFPCS were developed.

"Model CIM-Factories" As Learning Fields For ME Students

Hans-Jürgen Zebisch, Jörg Baumgärtner, Michael Seeland

Prof. Mag.sc.nat. DDI Bac.sc.hum. Hans-Jürgen ZEBISCH is Dean of the Mechanical Engineering Faculty at the Berufsakademie Karlsruhe and Head of the Division "Technical Computer Systems (TECS)" from the German HP User Group. Dipl.-Ing.(BA) Jörg BAUMGÄRTNER & Dipl.-Ing.(BA) Michael SEELAND are third party paid Assistat Engineers in his CIM-Laboratory; they are also members of the CSL-ME (Software/Macro Library for ME-10/30 Users) Team. Address: Berufsakademie Karlsruhe, CIM-Laboratory, Kaiserallee 11, D-7500 Karlsruhe 1. Tel. office: +49 (0)721 135-5114/5124; Tel. private: +49 (0)6257 5440; Fax.: +49 (0)6257 7541.

0. Introduction

The "Berufsakademie Baden-Württemberg (BA)" uses in the moment worldwide the largest installation of HP CAD Systems in the field of egineers education. This short presentation focuses on three information goals:

1) What is the structure of the BA's engineering education as a new approach to cooperative education?

2) What are the basic ideas of our CAx-training network, the so called "Model CIM-Factory"?

3) What happened with the BA's activities concerning the German HP Users Group and in point of co-operation with Hewlett-Packard?

1. Berufsakademie, an alternative to university courses

Figure 1 shows a typical problem of computer users; an interface problem. But such interface problems do not only exist between hardware components and between software applications. They also exist between an engineering education system - on the one hand - and an engineering employment sy-

fig. 1

stem - on the other hand. The Berufsakademie was developed in the state of Baden-Württemberg as a special center of vocational training and education for secondary school graduates, which closely integrates theory and practice and helps in opening up career opportunities comparable to those of university graduates. Employers praise it for the institutionalised job-orientation, in terms of a high level of competence and very low numbers of unemployment of graduates. *It's goal is to provide an engineering education with a minimum of such interface problems.*

The fast growing Berufsakademie has meanwhile been made a permanent feature of the educational system of Baden-Württemberg. Given this background, we have three paths of engineering education in this German state (**fig. 2**):

*   the traditional way: Technical Universities (TU - the only way to get a doctors degree),

*   the nearly 20 years old way: Fachhochschulen (FH - an evolution of the old engineering schools),

*   the nearly 15 years old way: Berufsakademien (BA - based on the so called "Stuttgarter Modell" developed by companies like DAIMLER-BENZ, BOSCH and SEL), established in order to provide an attractive alternative to existing institutions of higher education

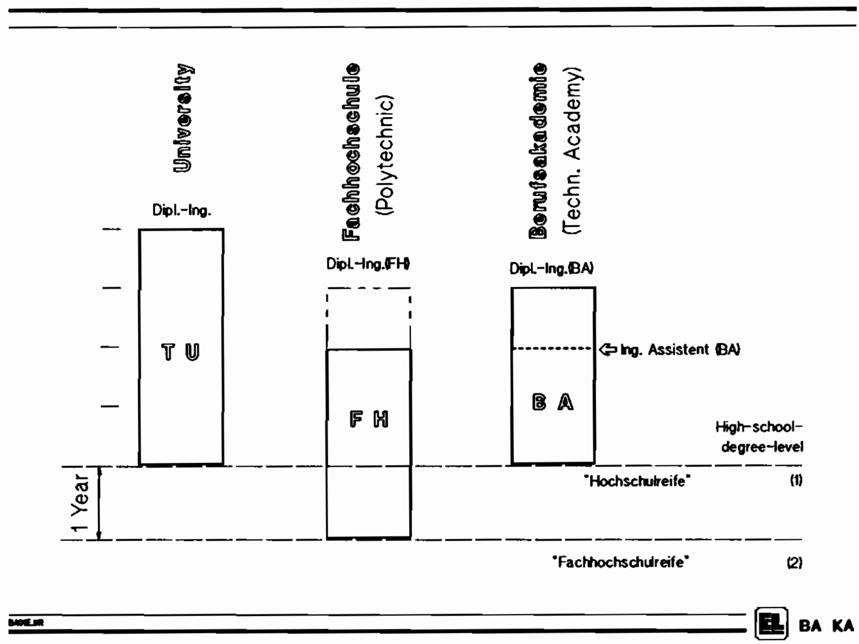


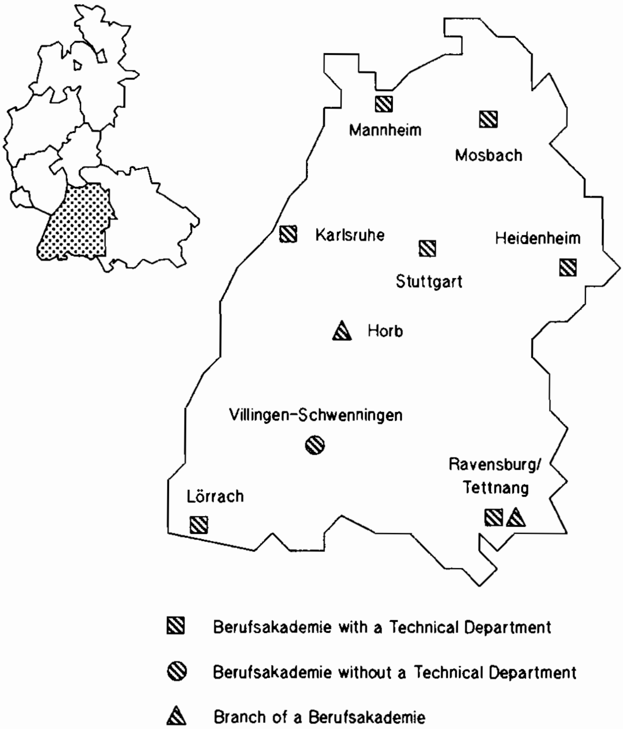
fig. 2

The "Berufsakademien Baden-Württemberg" (BA) are "Technical Academies" which require three years of studies leading to a degree in the field of Engineering [with several options] - Dipl.-Ing.(BA) (= Dipl.-Engineer of Berufsakademie).

The special feature of this plan of studies is a dual type of engineering-education, that means a very close co-operation between employers and the states goverment. Students who are enrolled in this program must be employed (as trainees - a training contract with standardized clauses governs the terms of traineeship) by a company located in the region of the partner-academy. Given this background, it is possible to say that both "dual-partners" (academy and employing-company) provide an engineering education that allows students to accumulate a lot of practical job experience.

It is possible to study mechanical-engineering (me) at seven academies in Baden-Württemberg: Heidenheim, Karlsruhe, Lörrach, Mannheim, Mosbach, Ravensburg und Stuttgart. Nearly 200 companies - mainly middle sized companies (less than 1000 employees) - are education partners of the academies in the me-field. The locations of the Berufsakademien in the state Baden-Württemberg are shown with the map in **figure 3**; they are well distributed over the land.

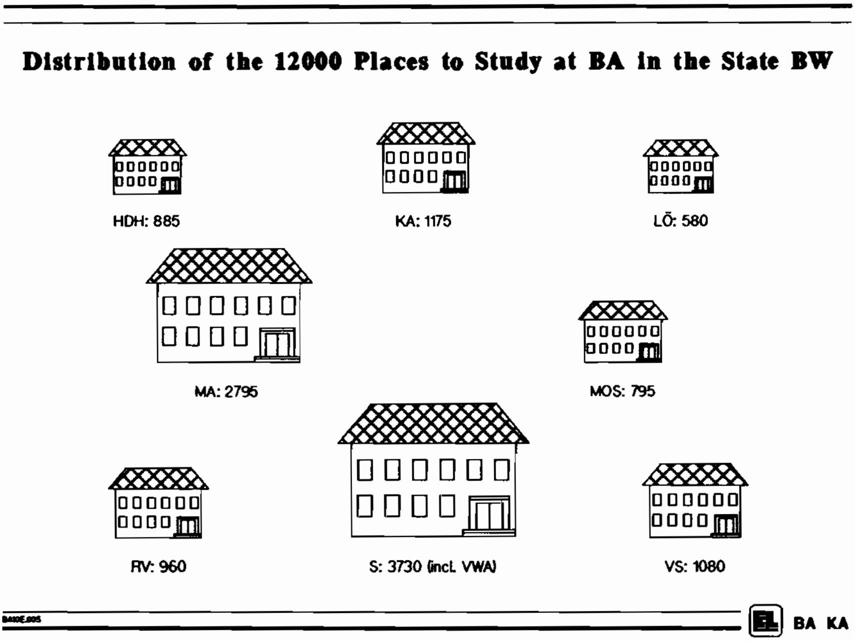
Locations of Berufsakademien in Baden-Württemberg


Mannheim
Mosbach
Karlsruhe
Heidenheim
Stuttgart
Horb
Villingen-Schwenningen
Ravensburg/ Tettnang
Lörrach
▨ Berufsakademie with a Technical Department
● Berufsakademie without a Technical Department
▲ Branch of a Berufsakademie

BANE.001 BA KA

fig. 3

An essential characteristic of these courses is the combination of special scientific knowledge and practical training: in alternating phases (dual system) the Berufsakademie provides science-related and work-related education at academies and training companies. Throughout the three-year training course, the student attends the academy and at the same time works as a trainee in a contractual training relationship with a company; alternating between theoretical and practical training phases. Each term is subdivided into two blocks of 12 weeks: one block of theoretical studies at the academy and one block of practical on-the-job-training. In



Distribution of the 12000 Places to Study at BA in the State BW

fig. 4

the course of his training the student has to take more and more responsibility and is partly integrated into the normal working process of his training company. Thus, the student's motivation is increased and the efficiency of the various learning processes improves.

The BA was founded in 1974. Fifteen Years later there are 8 academies with 12,000 (the distribution of the places to study shows the **figure 4**) sponsored students and 3,500 employers offering BA training facilities (Business Administration, Engineering, Social Work). In the *technical training field* the following fields of specialization (**fig. 5**) are offered:

* electrical engineering
* mechanical engineering
* conservation and radiation protection
* technical computer science
* timber and plastic technology

Those courses are subdivided into two stages, each of which ends with a state examination. The first two years (1st stage) lead to a first job qualification, the final qualifica-

Specialisation in the training field of Technology Technical Academy - Government approved Academy of Studies	Electrical Technology				Mechanical-Engineering				Radiation Protection			Technical Information Science		
	Automation Technology	Energy Technology	Communication Technology	Timber Technology	Precision Mechanics	Production Technology	Design	Process Engineering	Medical Field	Nuclear Power Plants	Environment Protection	Prod. Inform. Science	Data Processing	Telecommunication
Heidenheim						☆	☆							
Karlsruhe	☆	☆	☆		☆	☆	☆		☆	☆	☆	☆	☆	☆
Lörrach	☆	☆				☆	☆					☆	☆	
Mannheim	☆	☆	☆			☆	☆	☆				☆	☆	☆
Mosbach	☆			☆	☆		☆					☆		
Ravensburg / Tettnang	☆		☆			☆	☆					☆		☆
Stuttgart / Horb	☆	☆	☆		☆	☆	☆					☆	☆	☆

fig. 5

Structural Characteristics of BA

1 Work and science orientation

2 Alternating training in different places of learning

3 Integration of theory and practice

4 Structure of the study and training course

5 Different categories of teachers

6 Co-operation of partners with equal rights

fig. 6

tion is achieved after an additional year (2nd stage). Graduates in the technical training field are awarded by the state the following titles, stating in paranthesis the awarding institution:

* after the first stage: "Ingenieurassistent (BA)"
* after the second stage: "Diplom-Ingenieur (BA)"

The first certificate permits an early start into working life without throwing the student back to his point of departure, the "Abitur" (secondary school leaving certificate = high school degree level 1), in case the diploma of the second phase could not be obtained.

In order to reach a better harmony with work requirements and to establish close contacts between specialized science and professional practice, the BA received a different structure than universities and Fachhochschulen (polytechnics) from the very beginning. The structural characteristics are shown in figure 6.

The experimental phase of the Berufsakademie in the state Baden-Württemberg was started in 1974. It was modelled on the so-called "Stuttgarter Modell". The BA became a regular educational institution (4th of May 1982) by the law on Berufsakademien. As common bodies of all academies a governing body and specialized committees for the commercial, the technical and the social field were set up with the Science and Arts Ministry (fig. 7). The governing body adopts recommendations on all questions of fundamental concern.

On the state level, these specialized committees are responsible for all questions concerning their field of specialization. It is their duty to lay down the training and study contents for the theoretical as well as for the practical phases (objectives and substance). The governing body and the specialized committees are composed of an equal number of representatives of the state and of the training companies involved.

On the local level, the functions of the Berufsakademie are fulfilled by joint operation of academy and training companies. Each academy has a co-ordinating committee acting as a link between these two places of learning; academy and training companies are equally represented on this committee who admits training institutions after having satified themselves that they offer the required training facilities.

Research on training results and career trends of Berufsakademie graduates has shown that nearly all of these graduates emphatically stand by their former choice of training cour-

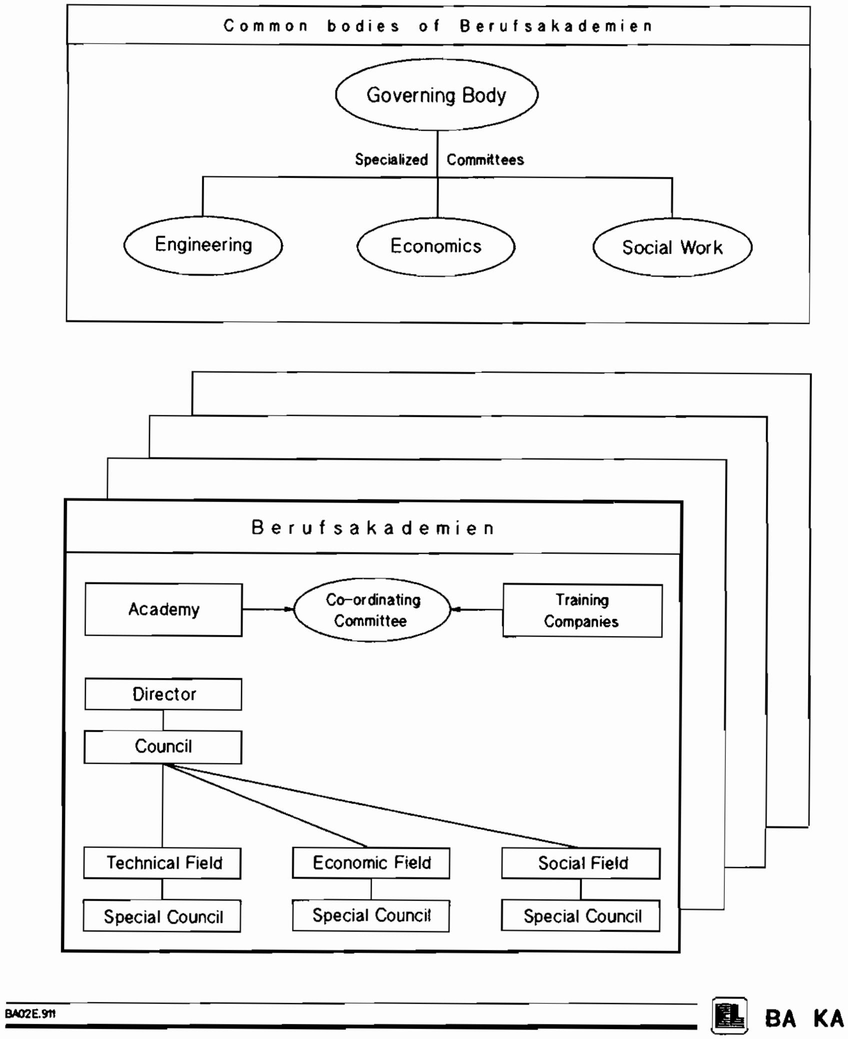
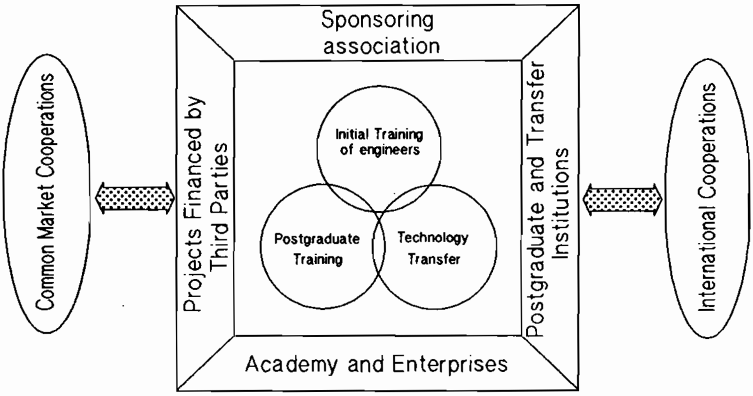


fig. 7

se. After completion of their training they work in qualified positions without any period of vocational adjustment, and about two years later most of them occupy positions as executive clerks, some of them even lower or middle management positions or managements assistant posts. One can notice that financially Berufsakademie graduates are largely on

Economical Sponsoring by a Network Structure of the Engineering Training System


Sponsoring association
Common Market Cooperations
Projects Financed by Third Parties
Initial Training of engineers
Postgraduate Training
Technology Transfer
Postgraduate and Transfer Institutions
International Cooperations
Academy and Enterprises

BAHE.sm

BA KA

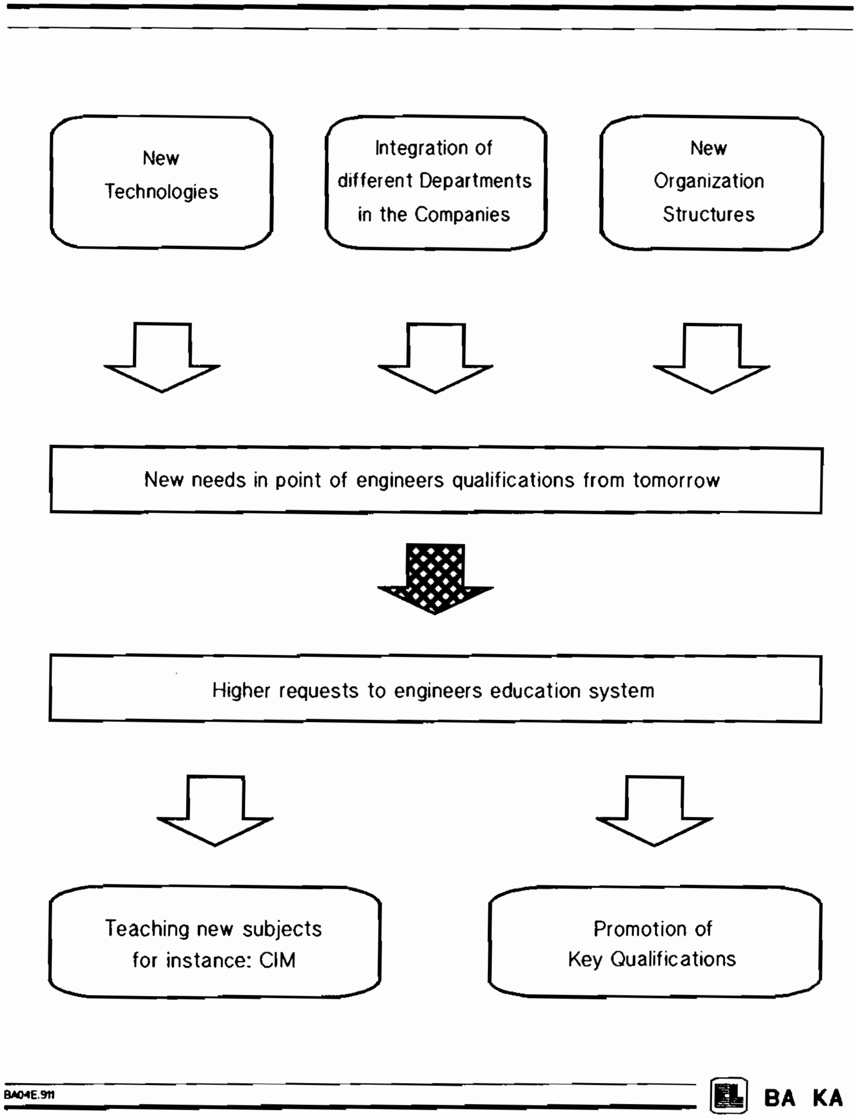
fig. 8

a par with college graduates, and their readiness for a change of employment is as big as that of university and college graduates. Their superiors almost unanimously emphasize their professional efficiency.

So far, so good! I hope this was an interesting overview on the structure and the educational activities of the Berufsakademie Baden-Württemberg today. It looks like that we will get other structures tomorrow, enriched with the sectors postgraduate training and technology transfer - embeded in a network of economical sponsoring (**fig. 8**). I also think it was a necessary informational background, because science-related and work-related CAx- and CIM-training for me-students needs more than PC's and workstations; it needs a coordinated curriculum and a practice-like training-equipment.

2. BA's educational goals in the field of CAx/CIM-training

To attain the mechanical-engineering qualifications of tomorrow, which are mainly defined by the "C-Technologies" like CAD, CIM ..., the curriculum for mechanical engineering



New Technologies

Integration of different Departments in the Companies

New Organization Structures

New needs in point of engineers qualifications from tomorrow

Higher requests to engineers education system

Teaching new subjects for instance: CIM

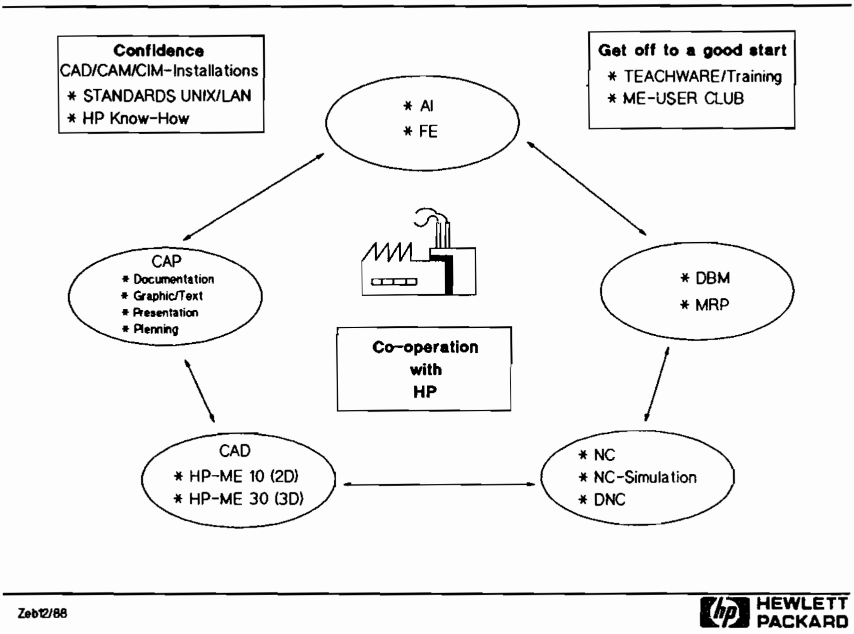
Promotion of Key Qualifications

BA04E.911

BA KA

fig. 9

education was extensively modified (for the typical structure have a look to **fig. 9**). Also, it was decided that CIM-laboratories (consists Computing-Lab, CAD-Lab, CAE-Lab, Tool-Machines-Lab, Robotics-Lab, Model Transportation and Storage Systems with professional controls) would be installed at each of the seven places of the academies with me-faculties.



Confidence
CAD/CAM/CIM-Installations
* STANDARDS UNIX/LAN
* HP Know-How

Get off to a good start
* TEACHWARE/Training
* ME-USER CLUB

* AI
* FE

CAP
* Documentation
* Graphic/Text
* Presentation
* Planning

* DBM
* MRP

Co-operation with HP

CAD
* HP-ME 10 (2D)
* HP-ME 30 (3D)

* NC
* NC-Simulation
* DNC

Zeb12/88

HEWLETT PACKARD

fig. 10

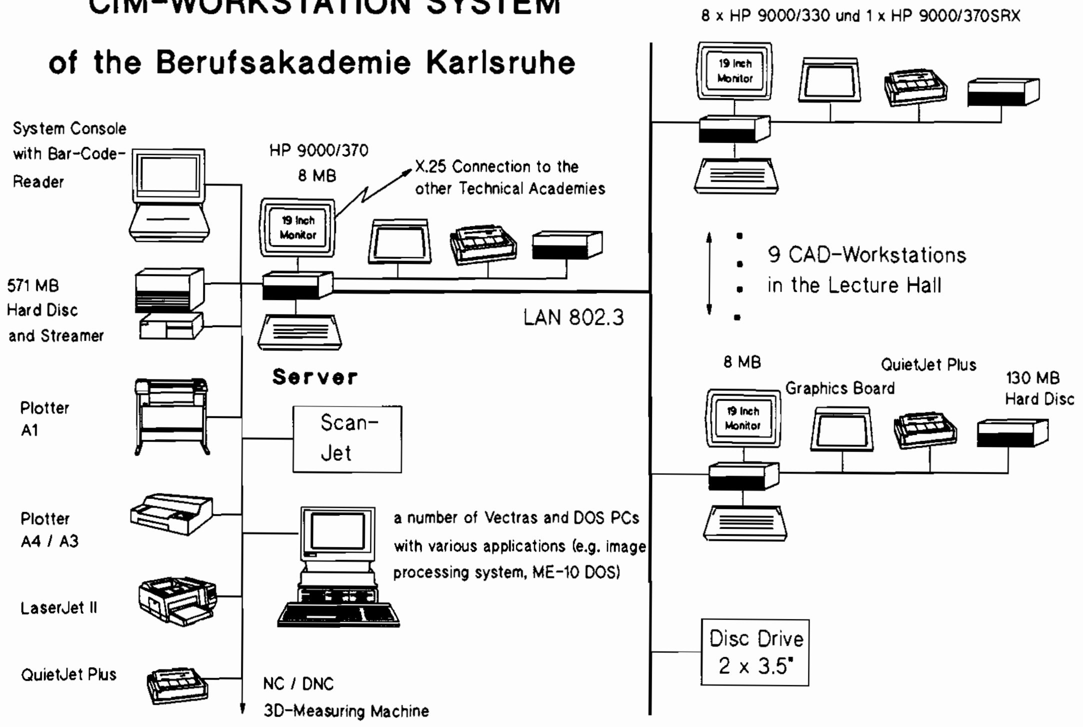
Within the first months a planning-board discussed this investment in a very detailed and comprehensive manner, and decided to install models of CIM-factories in every faculty of mechanical-engineering (HP's co-operation offer at that time is shown in **figure 10** - in this case it isn't unimportant, that user club activities in HP's opinion are a main argument of collaboration, and this brought us some workstations and a lot of software free of charge!). The goal: Students should find a well-equipped learning field that makes it possible to show and/or demonstrate the computer-integration of iland-solutions like CAE, CAD, CAM, CAP and CAQ all the way to computer aided logistics and which gives them the possibility to accumulate their own experiences in the CIM-field.

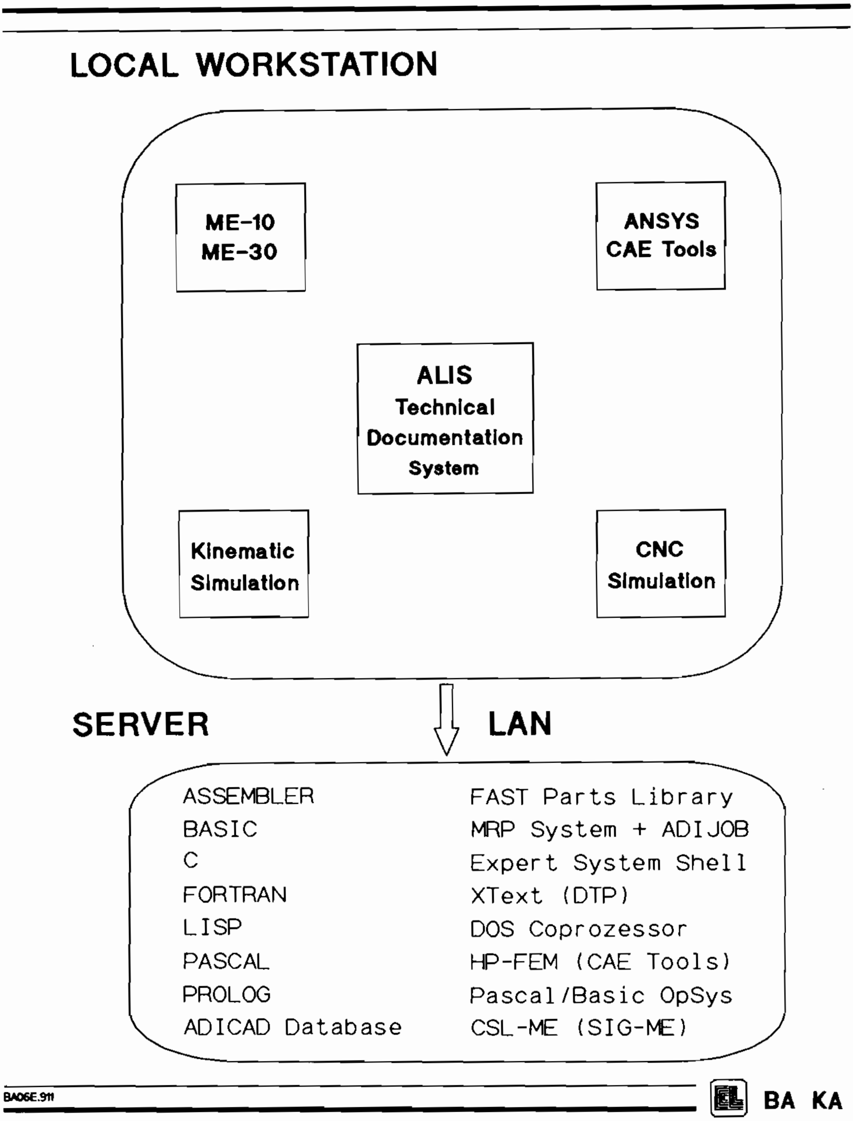
In the meantime - after a very strict selection process of offers from different vendors - we have installed a mixture of very important Hewlett-Packard hardware (**figure 11**) and software (**figure 12** - realization of an engineer's workstation from tomorrow as far as possible with software-tools from today), but in this case some problems will arise when comparisons are made with the traditional curriculum, because teaching the needed qualifications in the field of Compu-

fig. 11

CIM-WORKSTATION SYSTEM

of the Berufsakademie Karlsruhe





LOCAL WORKSTATION

ME-10
ME-30

ANSYS
CAE Tools

ALIS
Technical
Documentation
System

Kinematic
Simulation

CNC
Simulation

SERVER

LAN

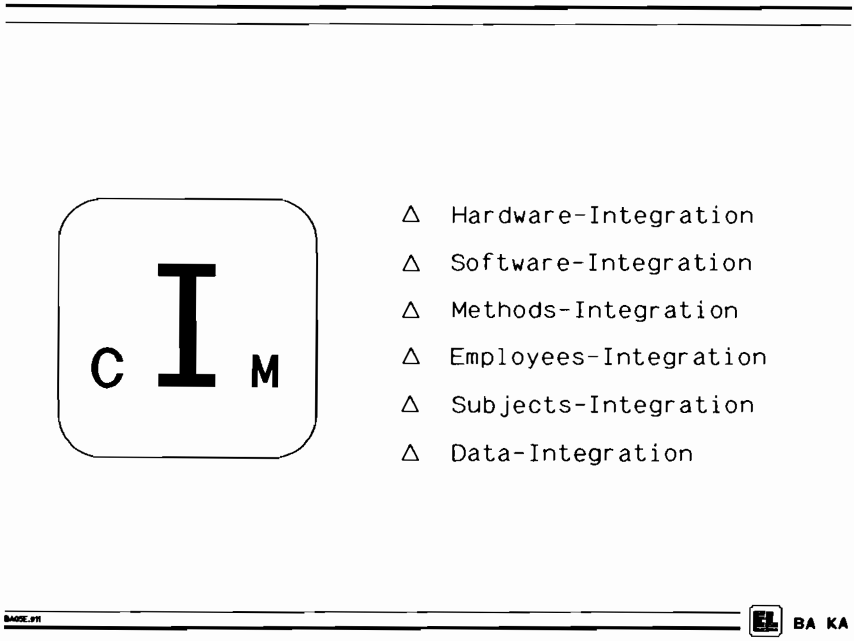
ASSEMBLER	FAST Parts Library
BASIC	MRP System + ADIJOB
C	Expert System Shell
FORTRAN	XText (DTP)
LISP	DOS Coprozessor
PASCAL	HP-FEM (CAE Tools)
PROLOG	Pascal/Basic OpSys
ADICAD Database	CSL-ME (SIG-ME)

BA06E.911

BA KA

fig. 12

ter Integrated Manufacturing requires a curriculum without borders between the special technical subjects; integrated learning arrangements (learning by working with projects, case studies, study works in the frame of inter-faculty project groops etc.) are necessary: CIM needs more then only hardware- and software-integration (**figure 13**).



- △ Hardware-Integration
- △ Software-Integration
- △ Methods-Integration
- △ Employees-Integration
- △ Subjects-Integration
- △ Data-Integration

fig. 13

We have to find new educational solutions in the frame of an educational-partnership. The goal is an interdisciplinary structure (curriculum) of engineering education in "Model CIM-Factories" (**figure 14**), as well as an initial testing of this way of learning and final evaluation. It is necessary to plan the CIM-training (included key-qualifications) very compactly, because there is no possibility to increase the number of lessons per week. Doing this gives us the chance to transfer the evaluated solutions into the areas of further education (postgraduate training). Also in the planning stages is the creation of CIM-self-education materials and CBT-Sessions (CBT = Computer Based Training).

The characteristic of the starting-step was to define and to install the necessary hard- and software, and also to create an optimal local network. Additionally we set up a network for communication and experience-transfer between all the me-faculties of the seven academies. On all the academies there exists an executive-manager for the whole CIM-equipment. They are all linked in the so-called CIM-working-group (founded by the states goverment). It is embedded in the structure which is shown in **figure 15**, and its main goals are to accomplish the following:

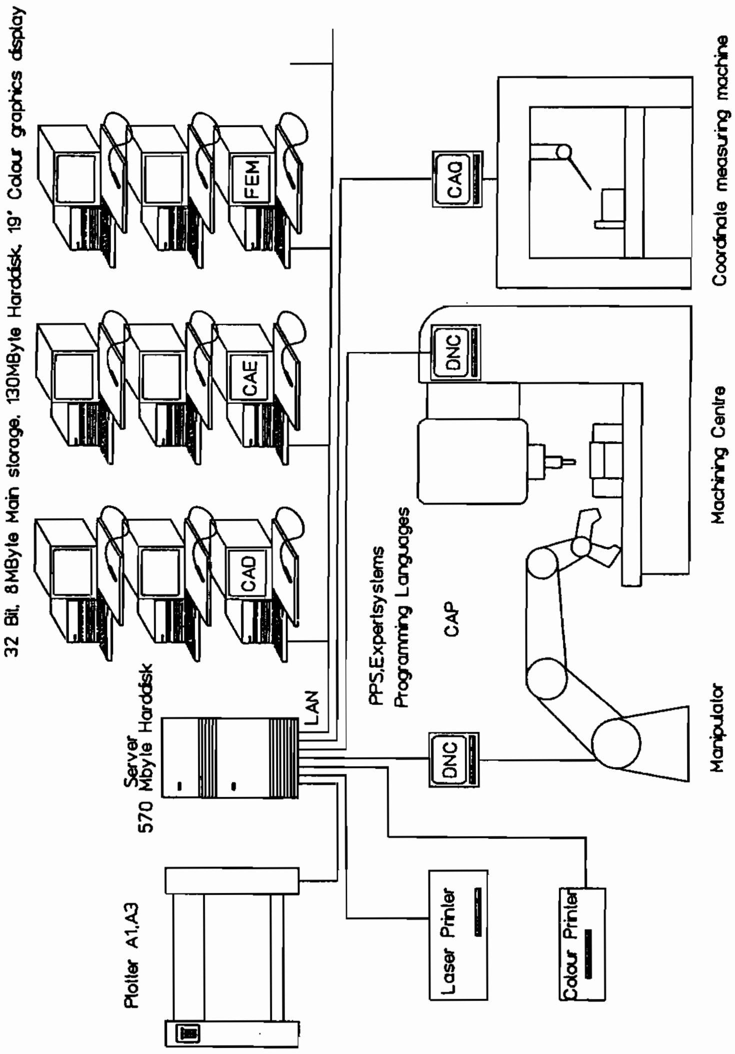

32 Bit, 8MByte Main storage, 130MByte Harddisk, 19' Colour graphics display
CAD
CAE
FEM
Server
570 Mbyte Harddisk
LAN
Plotter A1,A3
Laser Printer
Colour Printer
PPS,Expertsystems
Programming Languages
CAP
DNC
DNC
CAQ
Manipulator
Machining Centre
Coordinate measuring machine

fig. 14

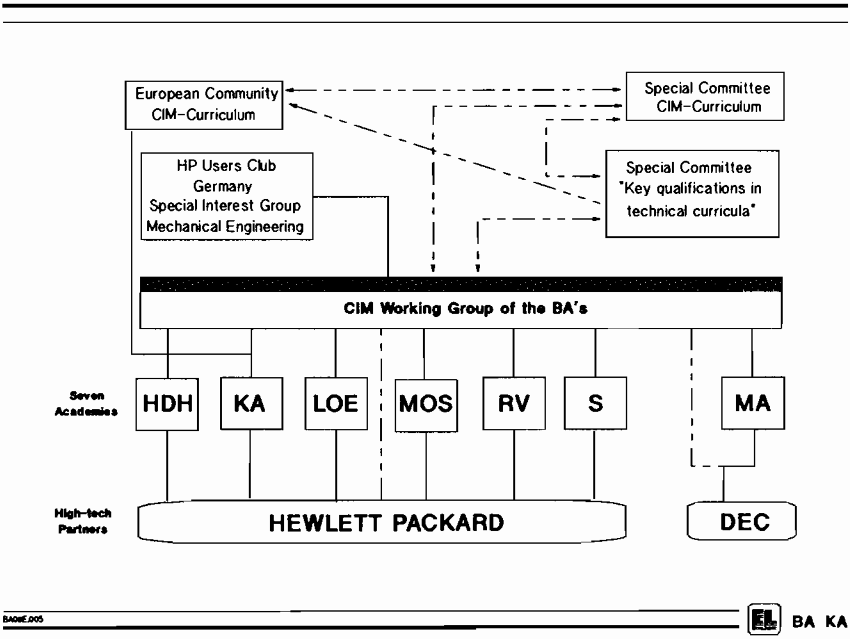

European Community CIM-Curriculum
Special Committee CIM-Curriculum
HP Users Club Germany Special Interest Group Mechanical Engineering
Special Committee "Key qualifications in technical curricula"
CIM Working Group of the BA's
Seven Academies
HDH
KA
LOE
MOS
RV
S
MA
High-tech Partners
HEWLETT PACKARD
DEC
BA06E.005
BA KA

fig. 15

- Knowing how to transfer installations and user problems,

- fulfilling the co-operation with the high-tech-partners DEC and HP,

- a hotline service (problem solving databases),

- interchanging project- and students-work-solutions,

- a news letter and other information services,

- developing software tools, training materials, teachware etc.,

- co-ordinating technology transfer academies -> industry.

In the meantime we installed not only a network of information interchange, we also created a data transfer network between the seven academies (**figure 16**). In fall (2nd and 3rd October) 1989 we held up the first symposium "using computers in the technical training field" - sponsored by Hew-

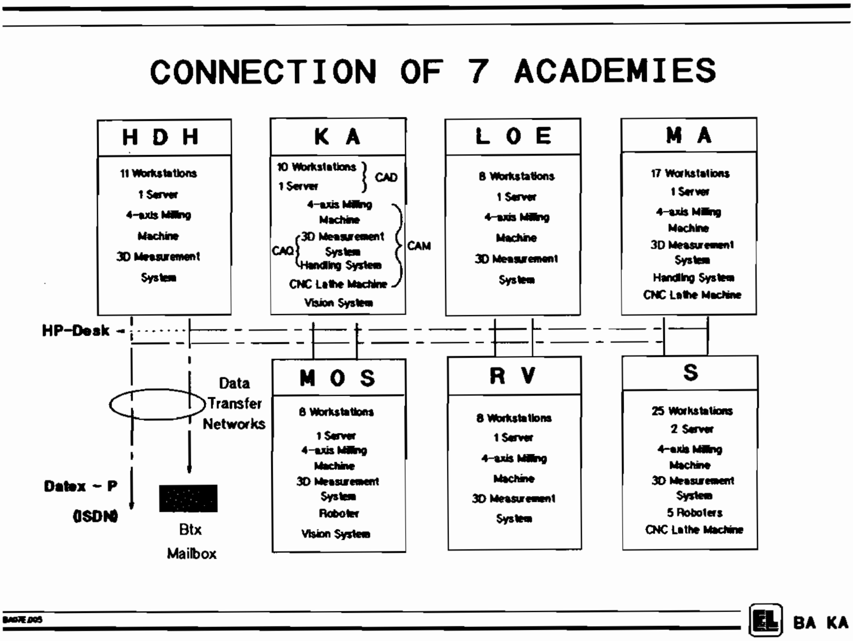


fig. 16

lett Packard, the German HP Users Group and the European Community - in Karlsruhe: an annual platform for information interchange between teachers in higher education.

3. BA's activities in the frame of the German HP Users Group

HP as our high-tech-partner was very interested that the academies should be active partners in the HP German Users Group. This was in addition to our own specific interest in point of a good working technology transfer. **Figure 17** shows the structure of the TECS-division in our users group. The basic-platform is given by the local meetings. In the state Baden-Württemberg they are all placed at BA's and the leaders of those local meetings are BA-professors. The idea of the CLS-ME is also born at BA's, and our students developed a lot of very helpful tools and software-packages in the last two years. **Figures 18 - 21** shows some CSL-ME Units. We will present them in action with ME-10 DOS as platform.

Our personal opinion: A model of co-operation who should be (modified) copied and sponsored by NUGs and Interex.

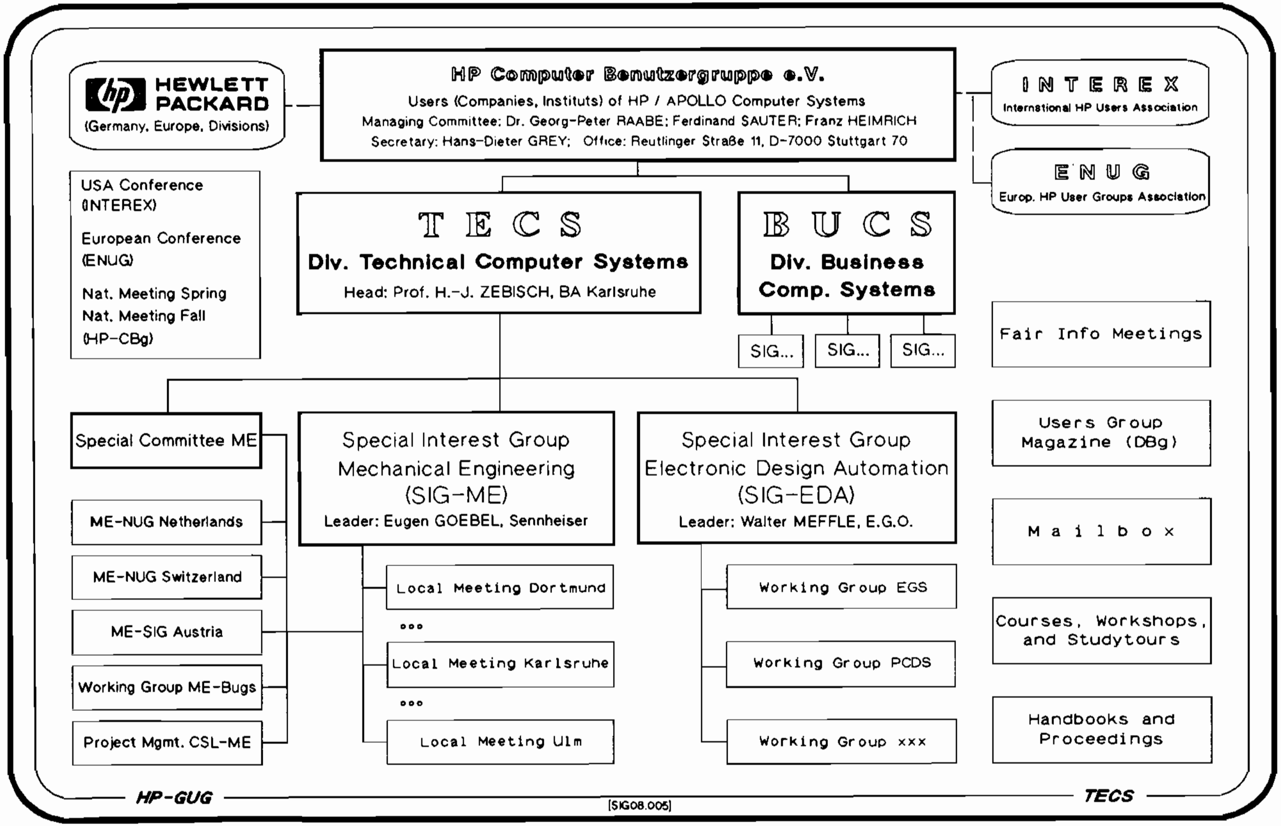
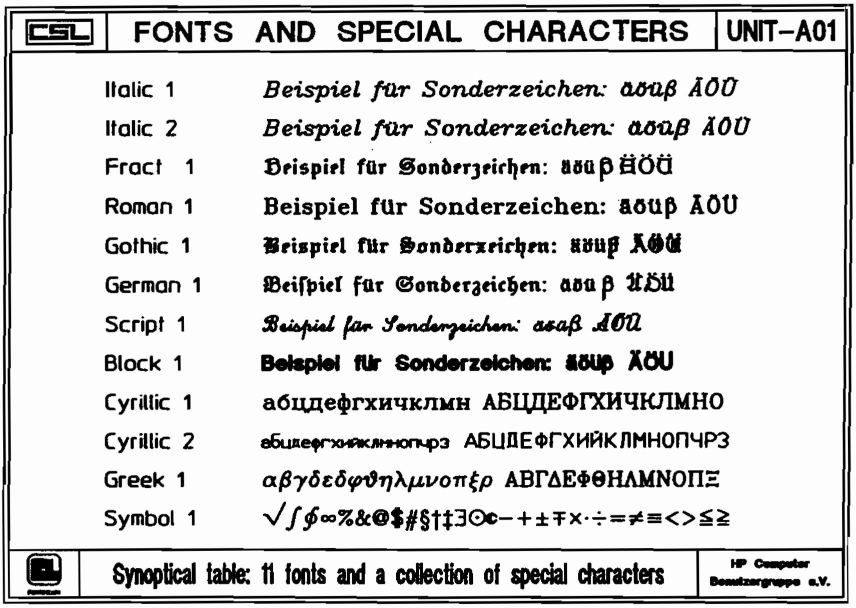

HEWLETT PACKARD
(Germany, Europe, Divisions)
HP Computer Benutzergruppe e.V.
Users (Companies, Instituts) of HP / APOLLO Computer Systems
Managing Committee: Dr. Georg-Peter RAABE; Ferdinand SAUTER; Franz HEIMRICH
Secretary: Hans-Dieter GREY; Office: Reutlinger Straße 11, D-7000 Stuttgart 70
INTEREX
International HP Users Association
ENUG
Europ. HP User Groups Association
USA Conference (INTEREX)
European Conference (ENUG)
Nat. Meeting Spring
Nat. Meeting Fall
(HP-CBg)
TECS
Div. Technical Computer Systems
Head: Prof. H.-J. ZEBISCH, BA Karlsruhe
BUCS
Div. Business Comp. Systems
SIG...
SIG...
SIG...
Fair Info Meetings
Users Group Magazine (DBg)
Mailbox
Courses, Workshops, and Studytours
Handbooks and Proceedings
Special Committee ME
ME-NUG Netherlands
ME-NUG Switzerland
ME-SIG Austria
Working Group ME-Bugs
Project Mgmt. CSL-ME
Special Interest Group Mechanical Engineering (SIG-ME)
Leader: Eugen GOEBEL, Sennheiser
Local Meeting Dortmund
...
Local Meeting Karlsruhe
...
Local Meeting Ulm
Special Interest Group Electronic Design Automation (SIG-EDA)
Leader: Walter MEFFLE, E.G.O.
Working Group EGS
Working Group PCDS
Working Group xxx
HP-GUG
TECS
[SIG08.005]

fig. 17



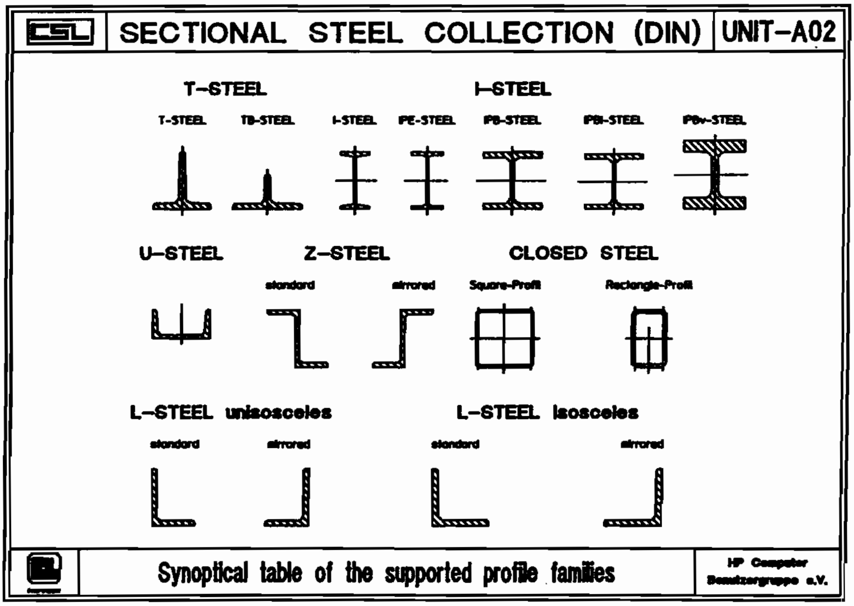
CSL | FONTS AND SPECIAL CHARACTERS | UNIT-A01

Italic 1	*Beispiel für Sonderzeichen: äöüß ÄÖÜ*
Italic 2	*Beispiel für Sonderzeichen: äöüß ÄÖÜ*
Fract 1	Beispiel für Sonderzeichen: äöüß ÄÖÜ
Roman 1	Beispiel für Sonderzeichen: äöüß ÄÖÜ
Gothic 1	Beispiel für Sonderzeichen: äöüß ÄÖÜ
German 1	Beiſpiel für Sonderzeichen: äöü ß ÄÖÜ
Script 1	*Beispiel für Sonderzeichen: äöüß ÄÖÜ*
Block 1	**Beispiel für Sonderzeichen: äöüß ÄÖÜ**
Cyrillic 1	абцдефгхичклмн АБЦДЕФГХИЧКЛМНО
Cyrillic 2	абцдефгхийклмнопчрз АБЦДЕФГХИЙКЛМНОПЧРЗ
Greek 1	αβγδεδφϑηλμνοπξρ ΑΒΓΔΕΦΘΗΛΜΝΟΠΞ
Symbol 1	√∫∮∞%&@$#§†‡∃⊙c−+±∓×·÷=≠≡<>≤≥

Synoptical table: 11 fonts and a collection of special characters

HP Computer Benutzergruppe e.V.

fig. 18

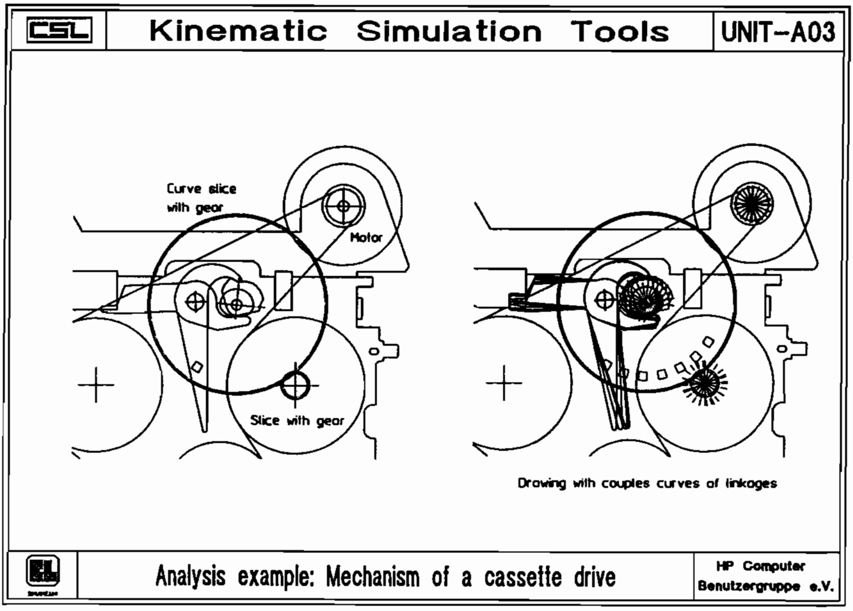


CSL | SECTIONAL STEEL COLLECTION (DIN) | UNIT-A02

Synoptical table of the supported profile families

HP Computer Benutzergruppe e.V.

fig. 19

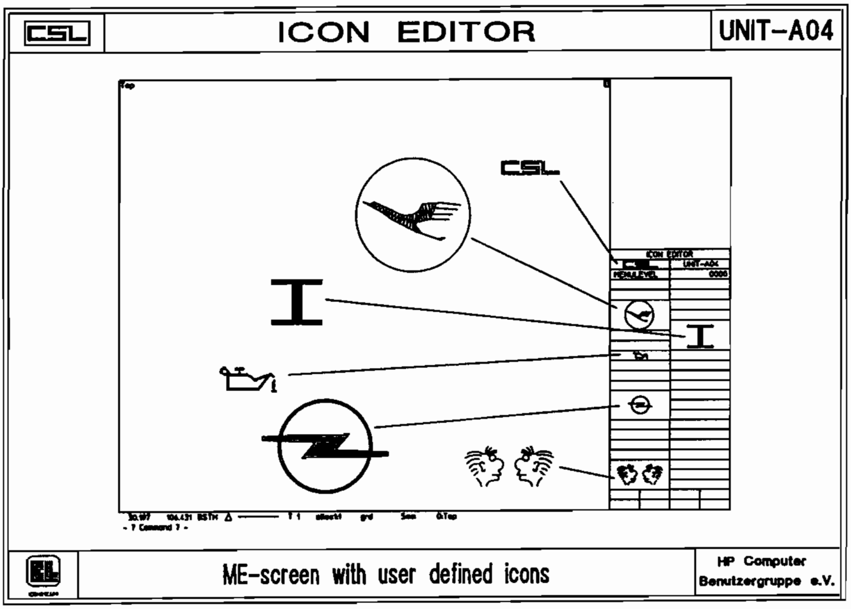


CSL | Kinematic Simulation Tools | UNIT-A03

Analysis example: Mechanism of a cassette drive

HP Computer
Benutzergruppe e.V.

fig. 20



CSL | ICON EDITOR | UNIT-A04

ME-screen with user defined icons

HP Computer
Benutzergruppe e.V.

fig. 21

PAPER TITLE:ME10/ME30 -AN EXAMPLE IN A NETWORKED ENVIRONMENT-

AUTHOR :Mr LUCIANO CALZAVARA

COMPANY :SECONDO MONA S.p.A.
Via Carlo del Prete,1
21019 Somma Lombardo
-ITALY-

phone (331)-256201

1 INTRODUCTION

The article details the decisional process used for the introduction of a CAD system within a medium size mechanical company.

It is described the decisional approach,the process undertaken to profile a specification,the possible suppliers selection,the final supplier individuation.

The chosen system runs on HP hardware.The computers are HP9000,300's serie models,while two HP1000's,A600+,are used as servers.

The software installed on the workstations is ME10 and ME30.

Generally, for a company which has a wide products range but in reduced lots, may not seem attractive to invest seriously in a CAD system, following the common guideline which states that such a mechanisation is appropriate only for repeatable designs.

The experience below described demonstrates instead that the advantages are of high profile also for a typical single shot projects environment.

As a first benefit a CAD organized design and drawing office allows an easy distribution of the workload between the engineers,and the way they work switches from the traditional sequential design to the more proficient concurrent design.

At a mature stage the individual results can be merged toghether for the design review and then, no matter which is the type or entity of the modifications introduced within the project itself, the design can be refined very quickly and efficiently without being involved in a confusing paperwork, error prone.Repeating the procedure with concentric loops as many times as required, the satisfaction for the resulting product is soon a rewarding benefit.

Many companies have problems to hire proficient design engineers.Until not long time ago, the engineers and draftsmen job at the drawing board was reputed not attractive because still padlocked to an old fashioned and static environment.It is difficult to believe that such engineers may eventually be involved in designing hightech products, e.g.machine tools, hydraulics and so on, but are still using barocque,time consuming, expensive and, why not?, boring work methods.

On diagram # 1 the average time spent by an engineer at the traditional drawing board is represented ,from start up to completion of a project.The worksharing is based on personal experience.

Consequently,the lack of interest of young people for the design engineer profession is not surprising.Thus the reduction of design engineers number, or if preferred, their constant number not mirroring the growing opportunities,is cause of concern.

Historically the industry policy is minded to production gains.

Such a trend has originated a huge demand of automation and, in turn, has returned to the industry itself an increased production capability. But, after a while, an unbalance within the same promoting industry itself became evident , because rougly the same number of traditionally geared design engineers has become unable to feed the fast growing demand of projects required to track the achievable production capacity.

Not to mention the requirements raised from the market, which acts as a brain squeezer,asking higher quality products in reduced times.Framed with the poor attraction of the design profession before mentioned,the picture is not comforting.

The situation can be better highlighted with the diagram # 2, representing two periods of time of the same typical company; the 1970's,with the shop floor equipped with traditional machine tools and the today production shop,with numerically controlled machine tools but still with a traditional design and drawing office behind it.

From the comparison,it becomes then clear that something has to be done to increase the design office throughput and to regain the control over the situation.

The choice is addressed to the CAD direction.

But two targets have to be focused first, before to outputting any financial resource:

a) CAD SYSTEM DESIGN

b) CAD SYSTEM CHOICE

2 REORGANIZATION

A CAD system can be essentially imagined either of distributed or centralized type.

A distributed system is based on more CPU'S.
A centralized system is typically host based.

Starting with the design of the CAD system, first above all, we have to fill the dream list.

We wish to introduce within our CAD system the following well defined conceptual points:

a) Centralized data base

b) Centralized back-up

c) Centralized software maintenance

d) Centralized software support to users

e) Centralized plotting

f) Drawing interchange capability with Customers having other CAD systems

g) Low training time

h) Unlimited user number and system scalability

2.1 CENTRALIZED DATA BASE

A centralized data base is essential, because users can have access to the last release of the authorized drawings without uncertainty.

Spreaded data bases are not recommended for the reason that the Company know-how must reside in a protected environment mainly for security concerns and not be easily available everywhere to everybody for everything.This protection is intended to be also physical, placing the data base in a controlled access room.

A tight real time control on the data base is a users interest too, because they know that they can rely on the system and consequently the archived files are uncorrupted and uptodated.

Under the cost profile the related hard disk stack, instead of being disseminated, is concentrated in a single site, then the solution is not affected by economical disadvantages.

2.2 CENTRALIZED BACK-UP

Mainly for the same security reasons, provisions has to be introduced for a centralized back-up system.A centralized service returns a high rate of safety to the overall system:

a) The operating system and data base back-up are done by an operator minded to the requirements, and then under convenient time schedules.

b) CAD station resident drawings back-up is done routinely and frequently, e.g.twice a week.In fact,CAD users tend to forget to periodically back-up their own work, and they realize the importance of such operation only when a file or directory has been lost for whathever reason.

Also with distributed CAD systems is economically convenient to have such a centralized back-up, because only a single unit is required instead than multiple. Thus purchase and maintenance costs are kept thin.

2.3 CENTRALIZED SOFTWARE MAINTENANCE

Whichever is the chosen CAD system, it is strongly recommended to have a remote seat capability to maintain the system aligned, without interruptions to the CAD users work.

With a distributed system it is not really efficient and practical to jog around the various CAD stations trying to update them;at least for the simple reason that editing the same file repeatedly may easily introduce errors or discrepancies.

The field experience has demonstrated that the system manager or related personnel physical presence at the CAD stations has to be kept at an absolute minimum ,otherwise the users, being not sensitive to the system tuning work necessary behind the curtains , may believe that the system itself is not reliable and in turn their work is not in the right hands.This reputation brings to an under use of the system and to a possible failure of the CAD introduction itself.

Also in this case an economical advantage can be easily seen.

2.4 CENTRALIZED SUPPORT

A mechanical or electrical design engineer for the execution of simple tasks has to achieve,who less who more, a minimum of familiarity also with the CAD operating system.

Then provisions are to be made available for supporting such users, preferably from a remote centralized seat.

2.5 CENTRALIZED PLOTTING

The plotting has to be carried out in a dedicated room or area, equipped with everything required to plot copies and transparencies in the desired size and number.

The plotters,no matter how many and of which type, are to be put under control of a single operator, which has the responsibility of all the system.The plotters concentration in a single room, equipped with a blue print machine and everything else required for hard copies production , has the goal of unload the engineers from routine tasks,leaving them more valuable time for the design. With this organisation,only one person will be responsible for the disposable materials like pens or/and toners, and for expensive supplies like paper and vellum.

Under the economical profile,a part the engineers time savings, a big improvement is determined by the de facto concentration of all the consumables in the hands of the plotters operator,who can merge all the material requirements, thus determining scale savings.

2.6 DRAWINGS INTERCHANGE

The capability of drawings interchange with other CAD systems may initially appear merely academic, but with the raising of number of installations whitin the industry, sooner or later it will be useful and necessary to be able to substain the electronic files interchange.

The advantages of this provision will be appreciated at the moment of loading complex drawings, when the time saved can reach the dimensions of weeks or months,thanks to the engineers of the partner company,who have practically done the job on our behalf.

The translator has to be IGES and not a simple plot file, because it is necessary to have as many information contained within the drawing itself available.

The economical advantage of such a capability is really impressive.

2.7 TRAINING TIME

The users training required is to be kept at low figures, for reasons of cost and for immediate system productivity.Indirectly,this can be a triple sided statement, because there are three different ways to answer the requirement:

-poor software;cheap,of easy use,really favourable learning curve but unfortunately with many holes,the dimensions of which may vary from package to package and are discovered too late.

-powerful software written by computer specialists for engineers.This kind of packages normally asks for a certain amount of computer skills which are not generally common to a standard mechanical design engineer.Then such a system asks for overspecialized design engineers;unfortunately,not too many of them are available .Leaving to these specialists the CAD installation cuts off the others,creating a questionable two speeds environment,whitin which at the end of the day what really counts is the average and not the top speed.From the company side point of view this concentration of power in the hands of few people is not recommended,evaluating the damage in the event of turn over.

-powerful software written,also,by engineers for engineers, thus with a familiar way of interaction and dialogue with the user, adopting his language and shape of mind.Computer skills in this case are not a compulsory requirement,thus achieving the target of requalifying all the involved engineers and, in turn, reaching a really satisfactory average constant speed and dampening in the same time the cost impact of the turn over.

2.8 NUMBER OF USERS

The CAD system must accept,theoretically,an unlimited number of users without becoming sluggish.At the same time, the system concept must be flexible enough to follow the future technology and to blend different suppliers togheter.

Always at the same time, the system must be scalable ,matching the professional level of each single user, without performance penalisation or oversized costs.

3 CAD SYSTEM TYPES

After having settled our basic requirements, we may start with the shopping, considering the three basic families of CAD systems currently available.

- PC based
- Workstation based
- Mainframe based

and bearing in mind which the size of the average file may be; in our case 750/1000 kbytes.

3.1 PCs

When we think PCs, we have to think low cost, otherwise,if the cost of the low end workstations is exceeded, there are no reasons of still considering the PC choice as a possible alternative.

Consequently,the advantages of a PC based CAD are mainly due to the relative popular cost and to the myriad of hardware and software suppliers available on the market.

Additionally,the system is easily integrated with other applications, such as word processing.Among the possible disadvantages for heavy CAD users are the lack of power and capabilities,poor 3D and generally absence of double precision accuracy.

Again, the hardware market is under continuous waves of new products following or anticipating different technical and marketing trajectories , and then may be difficult to expand the system in the medium period of time for the possible conflicts with the hardware or the software already installed.

The graphic engines are limited in power and it is also difficult to put together a system capable of working appreciably with a centralized data base, a centralized back-up, a centralized software maintenance and concurrent processes.

3.2 WORKSTATIONS

With the power of the available engines,a very wide span of applications may be covered . The performance of each workstation can be tuned for the specific task, thus we may believe that such a system can fullfill rougly all our requirements.But, still, the centralisation of such services like back-up, data base, plotting is not yet straight forward achieved.

The power of the available engines may satisfy or exceed the expectations of a broad spectrum of applications, from 2D to 3D solid modelling, with double precision accuracy.

Also in this case the rapidly evolving market may outdate the hardware, but choosing the operating system carefully will protect the workstations already installed,trailing them to the future.

3.3 MAINFRAME BASED

Being a centralized system it inherently offers a straight and clean solution for the general services like back-up or data base.On the other side it is heavily affected by the number and load of the connected CAD seats.

Adding power to such a system generally involves considerable financial resources and when the CPU becomes outdated,it may be difficult or impossible to keep the hardware and the software tuned to the always growing CAD requirements.

This choice can be imagined like a marriage with the manufacturer and may become dangerous, because the customer is subjected completely to the manufacturer wills and marketing policies.

Again, the number of users idle caused by a system fault has a certain financial weight.If the user wishes to prevent such inactivity, an increase of the running costs has to be accounted for,as determined by the major resources required for more responsive maintenance contracts.

4 THE RESULTING CAD

The ideal installation for our requirements seems to be workstation based, with all the machines tied together through a LAN and with UNIX as operating system.

For data base purposes and the other centralized services, a server computer gathers all the final drawings,thus keeping the hard disk size of each workstation to a reasonable dimension.

From the server,the operator or the system manager using TELNET can do a remote login to each workstation, without disturbing or interrupting the engineers at work.

The efficiency of the software maintenance and of the support to the end users is,in this way,greatly enhanced.

The workstations back-up is also carried out using the LAN itself. With a multiple tape cartridges unit, the routine back-up can be done automatically, using command files scheduling at fixed week days and time the back-up of the drawings contained within each workstation.Also in this case, the engineers work is not interrupted.

The plotters and the necessary pheripherals are wired to the server.When the CAD users need to plot,they send the file or files to be plotted to the server through the LAN. A command file scheduled from the CAD menu can thus instruct the workstation to send multiple files in batch mode,relieving the engineer,who may then follow his job whitout being minded on what is currently happening underground.

The server is able to detect whether the incoming file or files are only hard copies,required for any reason,or if they are the final releases.If the files are final releases,the server waits for the ASCII file originating the requested plot file from the workstation .If the match has been successfull,the server will then provide to update the data base and to divert the plot files to the plotters cue for the official transparency copies.

For a cluster of workstations located to a certain distance from the plotters room,it may be helpful to have an alphanumeric terminal locally ,on which the local CAD users can follow the plotters cue status.

5 SUPPLIER SELECTION

After having defined the system, the big task is the supplier selection.Few, but firm points, have been focused. In order of importance:

1) the h/w and s/w must be of the same manufacturer

2) the supplier must be on the market from long time

3) must be reliable

4) must have a good software support

5) must have a good hardware support

6) must bring sinergies

Point 1 is easily explained; when the user encounters a problem and is questionable whether the trouble is caused by a hardware fault or a software bug, he does not want to become the tennis ball between two suppliers, each one giving the other the problem responsibility.With this case, the lowest risk that a user may experience is the big amount of time spent, whrestling with two suppliers, trying to fix the problem.

The highest risk eventually encountered is an overall system malfunction, with performances not matching the promised expectations.As a resulting consideration,choosing the same supplier normally means to the user that the hardware performances are used at their best from the software, thus optimizing the system throughput.

Points 2 and 3 walk toghether.Investing a considerable amount of money is not like following the fashion trends.
The investment must work now and must survive.

Everybody realizes that ,at last,nobody knows the future, both of his own Company or of the Supplier, but looking to the past hystory of the Supplier may be a good pointer for metering his reliability. Long hystory normally means sound customer policy; like investment protection,good products reputation and capability to understand, to meet and govern the technical trends.

Points 4 and 5 also walk together.A non productive system for any reason, a software bug or a hardware fault, has to be recovered as soon as desired by the customer who has signed the relevant support contracts. A lack of response,or a response level not meeting the expectations,is a bitter experience,which inflates the running costs and lowers the system productivity.

Point 6 may seem the less interesting, at a first glance, but thinking of it more deeply may modify our opinion.

On the initial section of this presentation few basic concepts have been illustrated about the lack of skilled engineers and the needing of the industry for an interdisciplinary professional background.This requirement is particularly true for the mechanical engineers, who are generally less familiar with electronics and related disciplines.

Putting in their hands a CAD station is an "open-your-mind" operation,because they may familiarize with electronics on their own play ground.But this one is only a way that adds few points to the user qualification.If we are looking also for sinergies, the choice of the supplier is really of outstanding importance.

If we select a supplier not only for the CAD application, but,widening the view angle,also for sinergies, the final selection points a supplier able to become a partner in other fields like robotics,systems control, data acquisition.

Then the time spent on a satisfactory workstation for an apparently one-way CAD application, may induce the now more or less familiar user to employ the same scalable product on his own designs.

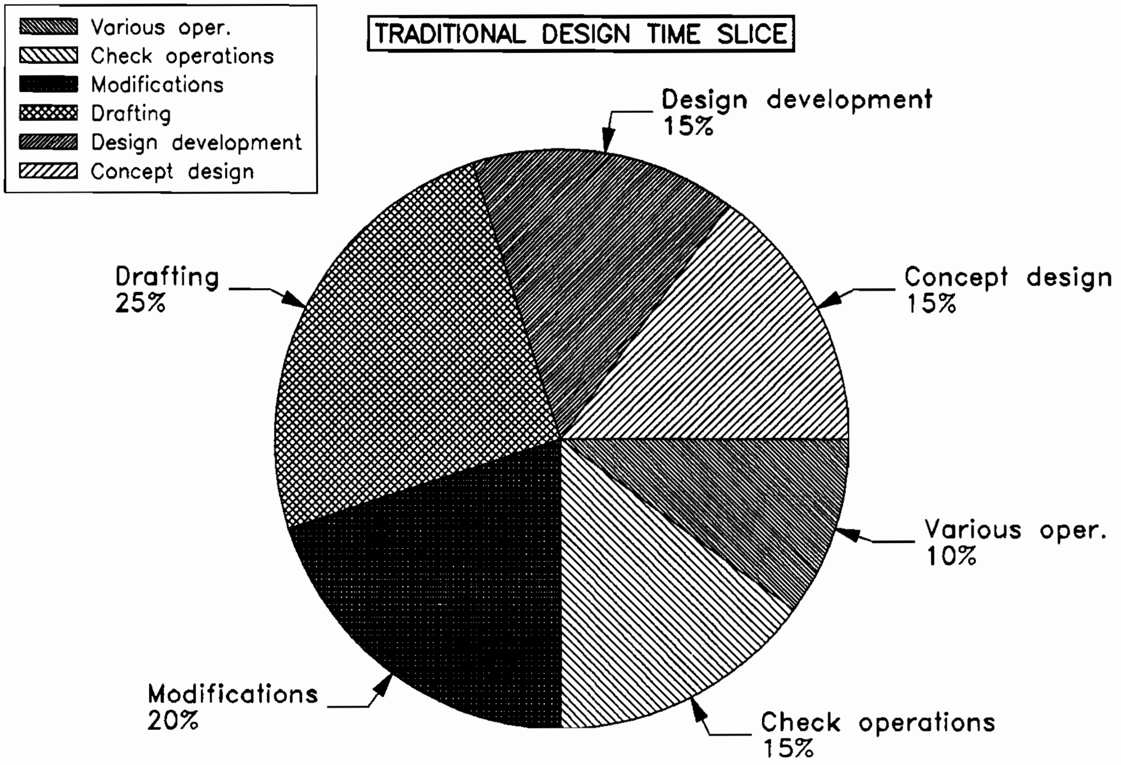
6 FINAL CONSIDERATIONS

The gain of time using the described CAD installation, expressed as an average value for single shot projects,is represented with the diagram # 3.

With this installation we have not yet erased the gap separating the design department output from the production floor capability,because a further 5% of difference still has to be recovered.

This 5%,or more, can be recovered reducing the 'Concept design' and the 'Design development' shares.Introducing CAE, at least a finite element analisys procedure,may improve the resulting figure,finally zeroing the difference previously reported.

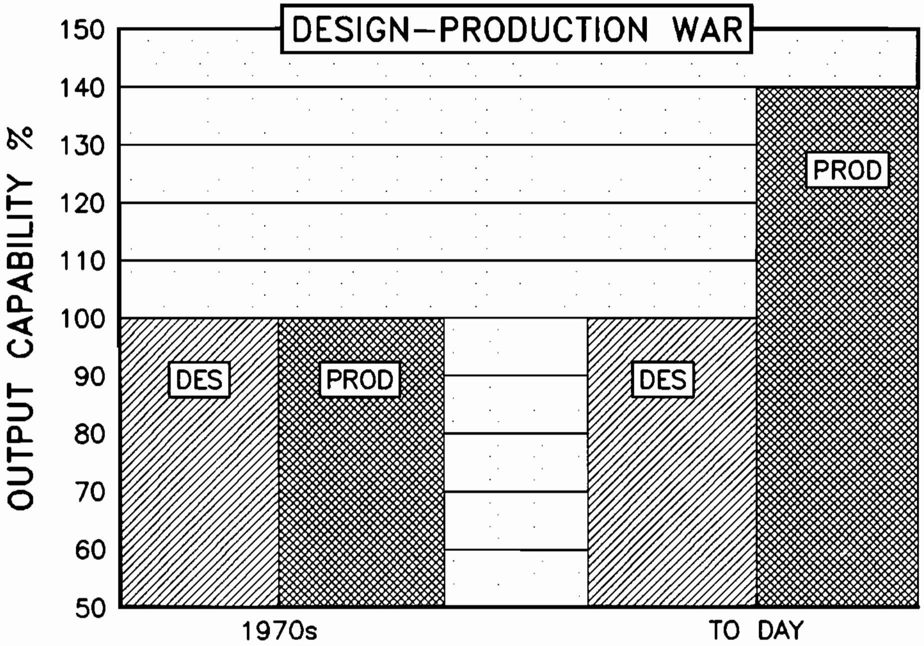
Diagram #4 returns the training times experienced with the chosen CAD system.



ME10/ME30 –An example in a networked environment–

1005–13

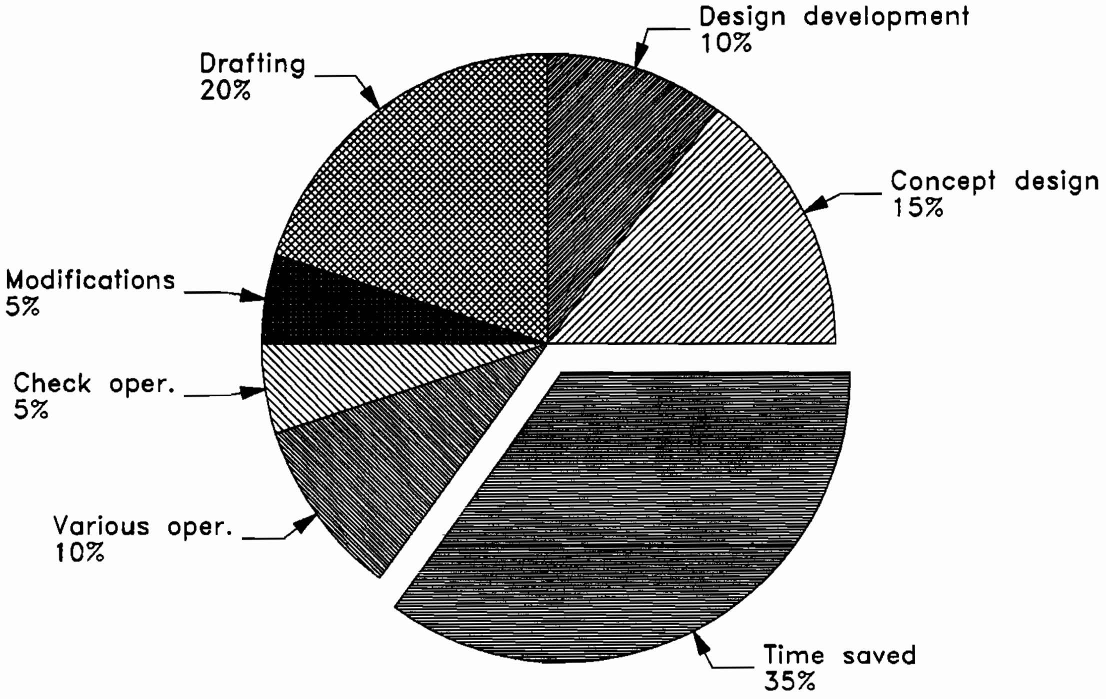
DIAGRAM #1



ME10/ME30 -An example in a computer environment-

DIAGRAM #2

CAD DESIGN TIME SLICE



ME10/ME30 -An example in a networked environment-

1005-15

DIAGRAM #3

CAD SYSTEM

LEARNING TIMES

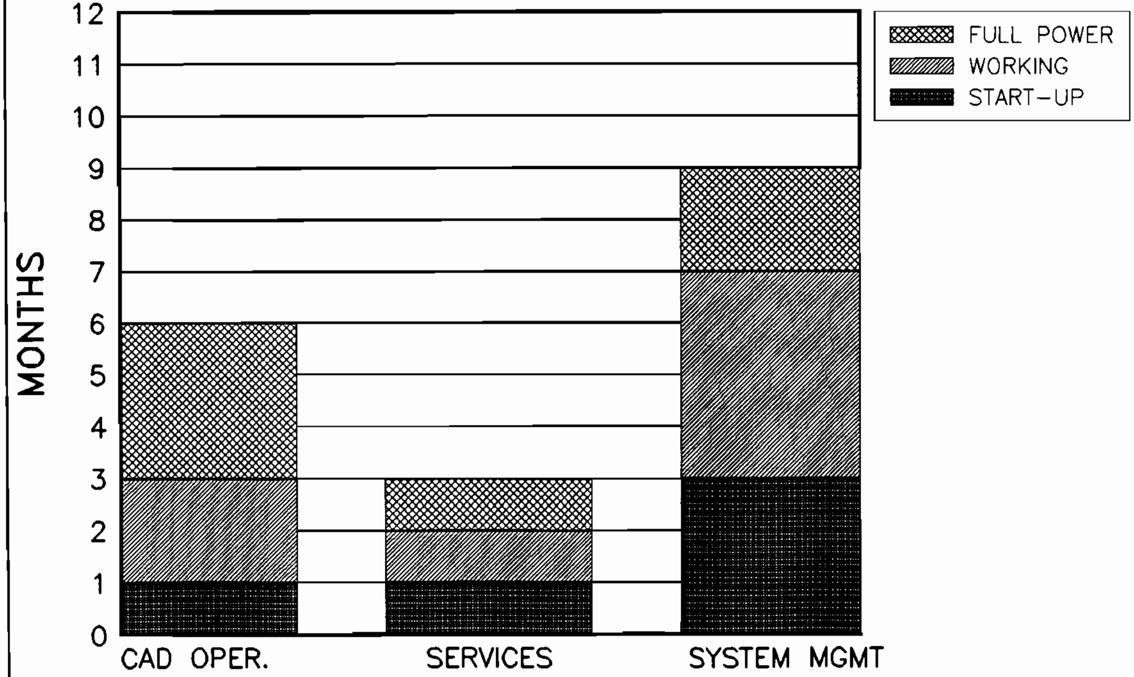


DIAGRAM #4

MUMPS - More Than a Childhood Disease And HP's Got It

George Hall
HBO & Company
301 Perimeter Center North
Atlanta, Georgia 30346
(404)393-6000

INTRODUCTION

"Languages have a deep relationship to the thought patterns of their users. If a programmer can't easily say something in a computer language, he is not inclined to think it, either. When programmers of different languages meet, they are projecting their thought patterns into each others' language. Finding that the language does not express these thought patterns as richly as their native language, they often judge the other language as inferior."

Thomas Munnecke, "A Linguistic Comparison of MUMPS and COBOL," Proceedings of the National Computer Conference, 1980.

It would be beyond the scope of this paper to attempt to make anyone into a MUMPS programmer. The purpose of the paper is to expose the systems analyst, technical manager or anyone who may need to make a technology selection in the future to the architecture of the MUMPS language, its strengths and weaknesses. Anyone in the data processing industry can benefit from exposure to all sorts of available technologies. MUMPS is a very attractive option for developing commercial applications on a UNIX system.

HISTORY and BACKGROUND

In the mid 1960's a consulting firm was trying to develop one of the first on-line hospital information systems at Massachusetts General Hospital with funding from the National Institutes of Health. This system, which was being developed in assembly language was to be accessed over phone lines by Teletype terminals throughout the hospital. Neil Pappalardo, who managed a group of programmers at the hospital became frustrated with the length of each cycle of the prototype process. About this time he had his first contact with an on-line high level language (called JOSS) and decided to write his own with his colleague Curt Marble. They started their work by borrowing some time on a PDP-4 DEC had donated to the psychiatry department. By 1966 they had named the first version Massachusetts General Hospital Utility Multi-Programming System (MUMPS). After some initial resistance the merits of the language were found to be compelling and development work using MUMPS began. The early spread of the language was by informal contact and word of mouth until 1969 when the developers left Massachusetts General to start their own companies to provide MUMPS based applications commercially. The development of MUMPS under a government grant allowed it to be in the public domain from the start and about 10 dialects of the language had emerged by the early 70's. Staff members of the National Institutes of Health realized that this diversity would be an obstacle to sharing the applications they were now funding at several institutions. Grants were given to the National Standards Bureau to oversee the development of a MUMPS standard and the MUMPS Development Committee (MDC) was formed. Money from government agencies and Massachusetts General were also used to establish the Mumps Users Group (MUG) to encourage the sharing of applications. By the mid 70's sister organizations were functioning in Europe and Japan. In 1977 the American National Standards Institute adopted a standard for MUMPS (national standard X11.1-1977). It was the fifth language to have an ANSI standard. What MUMPS needed at this point was a big break to start gaining broad acceptance in the academic and commercial communities. This break came in 1977 when the Veterans Administration, which operates the largest healthcare system in the free world hired a manager which was familiar with MUMPS through his previous job at the National Standards Bureau.

MUMPS has always had a public relations problem due in some part to the fact that its market is dominated by VAR's and OEM's. These vendors sell their applications and not MUMPS. Most MUMPS users don't know what technology they are using. Trade journals and reference books list and catalog all sorts of languages that never amount to anything while ignoring the large MUMPS install base and the many strengths of the language. During the early days of MUMPS application development at the VA, a jealous group of managers who were developing applications in COBOL made a "power play" of sorts and managed to have many of the MUMPS programmers and managers fired. They had many of the computers of the remaining ones confiscated and put in a warehouse. The MUMPS group continued to develop these applications without the knowledge or consent of the VA, calling themselves the MUMPS Underground Railroad. The National Association of VA Physicians, realizing the quality of the new applications took up their cause. An act of congress finally removed funding from the COBOL group and allocated the money to the distribution of the applications that had been developed by the MUMPS group. These applications formed a large core of public domain software, some of it healthcare specific and some of it general which have continued to be a strength of the language.

The MUMPS Users Group (MUG) estimates that there are over 47,000 MUMPS installations worldwide with about 70% of them in medical applications and 30% other applications in the Unites States. In the rest of the world the percentages are reversed. Other applications include accounting, inventory management, educational systems, word processing, telemarketing and artificial intelligence. It is used in more than 80 countries. The large amount of public domain software and the high degree of portability have made MUMPS the language of choice recently in many eastern block countries as they seek to find low cost ways to modernize industry and government. MUMPS' data management system (more on this later) make it an appropriate choice for commercial applications which need a large dynamic data base with frequent transaction type access.

MUMPS is available on all Hewlett-Packard hardware platforms that run under HP-UX, MS-DOS and XENIX. There are no immediate plans to port MUMPS to MPE/XL until a VAR of sufficient size and capability commits themselves to the platform. Previous attempts to port MUMPS to MPE/V were unsuccessful due in large part to the program address space limitations of the architecture. The amount of extra data segment swapping needed to keep the GLOBAL pointers and other necessary information available at memory access speed was excessive.

THE LANGUAGE

MUMPS can best be described as a 3 1/2 generation language. It is an interpreted language with a small number of very powerful commands and an imbedded data management system. This eliminates the need to know a Data Base Management System, Job Control Language, Linkage editor, System Definition Macros and programming utilities (sort/merge etc.). It is however a procedural language and it lacks a data dictionary, which would disqualify it as a fourth generation language by some definitions. Data dictionaries are available as a proprietary extension in some MUMPS implementations and the MUMPS Development Committee is working on a dictionary standard for the language.

VARIABLES/ARRAYS

MUMPS does not require the programmer to pre-define a variable or array, or to assign a type to them. Sparse arrays are used in which only the array elements that have been assigned a value occupy space. They may be subscripted to any arbitrary length with numbers or strings. For example the command:

```
SET PATIENT(123,900821,"HR")=""
```

stores the heart rate of patient 123 as taken on August 21, 1990. String subscripts reduce the need to use "keys" and tables to associate numeric subscripts with real world data. Arrays can be expanded dynamically without redefinition or affecting the organization of data already in the array. For example, you could go back later and decide to keep a previous diagnosis of patient 123 along with the date of the diagnosis by entering the following command:

```
SET PATIENT(123,"MEASLES",890601)=""
```

FILE SYSTEM/DATABASE

One of the strongest advantages of MUMPS is the imbedded database management system. It is so well integrated that there is very little that needs to be said about how to use it. You simply put a caret (up arrow) in front of a simple variable or an array specification and you are on disc. There is no need to declare buffers, open or close the file, issue read or write commands, determine the length of the media record or specify how the file is stored on disc. Those who are used to calling external data access routines or coding complex file specifications will be disapointed. These variables on disc are called GLOBALS and they are treated the same as local variables in every way. All of the features mentioned above under the VARIABLES/ARRAYS section, and below under the FUNCTIONS section go from "nice" to "remarkable" when you realize that they also apply transparently to disc files.

MUMPS GLOBALs incorporate many of the best and most productive advantages of relational databases, including the ability to change the way your data is related or used "on the fly" without lengthy database reorganization, and the ability to use the intuitive "row and column" model to understand your data. It is close enough conceptually to Structured Query Language (SQL) type databases that several third party packages are available which will turn SQL commands into MUMPS commands and allow SQL access to MUMPS data. GLOBALS are kept in a B-TREE type file structure. Different MUMPS implementations optimize the B-TREE in various ways.

PORTABILITY

It is almost a commercial necessity for a modern software language or database system to claim to be portable. Those who have really ported an application to a different hardware platform will attest to the fact that "portable" is a relative term. MUMPS is among the most portable of languages.

All of the early implementations of MUMPS included an entire operating system. This allowed MUMPS to handle multiple users efficiently on mid 60's hardware that was not supposed to be able to handle multiprocessing. When the MUMPS standard was developed many of the features which would normally be considered part of the operating system were written into the standard. This accomplished the standardization of elements that usually cause the most trouble in porting an application, including disc I/O, database management and terminal handling. In more modern implementations where MUMPS runs on top of a hardware vendor's native operating system the MUMPS interpreter is usually a large program which isolates the user from the native operating system to some degree and handles these issues in a portable fashion.

MUMPS COMMANDS

- o SET - Assigns values to variable locations.
- o KILL - Removes a variable from the list of variable names.
- o OPEN - Opens a terminal, printer or line printer for I/O.
- o CLOSE - Relinquishes control of a terminal, tape drive or line printer when I/O is completed.
- o USE - Defines the current device for the READ/WRITE commands.
- o WRITE - Directs I/O to terminals, tape drives and printers.
- o READ - Requests input from the user.
- o IF - Provides conditional branching. Places the truth value in a system defined variable called $TEST.
- o ELSE - ELSE is a separate command from IF in MUMPS. It evaluates a boolean expression and if the value is FALSE it executes the following command. If there is no expression to evaluate it checks the value in $TEST which was set be the last IF command.
- o GOTO - Unconditional branching.
- o DO - Branches to a referenced line or label, after which control is returned to the next line.
- o QUIT - Returns control to the command after the controlling DO command.
- o HALT - Terminates execution and returns control to the operating system.
- o HANG - Suspends execution for a specified number of seconds.
- o FOR - Performs controlled looping.
- o JOB - Causes commands to be run in background (batch) mode.
- o LOCK - Allows a user to lock a data item. The lock command gives the requesting user control of a logical resource. For example, LOCK ABC will give the user control of the name "ABC". If software conventions are not carefully followed, locking will mean nothing since there is no physical relationship between the "locked" resource and the data item.
- o BREAK - Suspends execution of the program to allow interactive debugging.
- o VIEW - Allows the programmer to dump portions of memory for debugging.

MUMPS OPERATORS

ARITHMETIC OPERATORS

+ addition
- subtraction
* multiplication
/ division
\ integer quotient division
# modulo, or remainder division

RELATIONAL OPERATORS

o STRING RELATIONAL
= equals
[contains
] follows
_ concatenate
? pattern match

o ARITHMETIC RELATIONAL
< less than
> greater than

LOGICAL OPERATORS
& and
! or
' not

SPECIAL VARIABLES

MUMPS defines several variables which are maintained by the MUMPS interpreter. They always start with the "$" character and they may be referenced by any program. Some of them are:

o $X - Contains the position of the cursor on the current line.
o $Y - Contains the number of lines from the top of the screen to the current cursor position.
o $HOROLOG - Contains the system date/time in a format which is part of the MUMPS standard.
o $TEST - Contains the truth value of the most recent IF/ELSE command.
o $IO - The current output device.

FUNCTIONS

The MUMPS standard defines a number of functions that provide for advanced string and numeric manipulation. Function names always start with the "$" character. Some of these functions are:

- o $ASCII - Returns the ASCII code number for a given character.
- o $CHAR - Returns the character representation for a given ASCII code.
- o $DATA - Returns a code telling you if a given element in an array is defined and if it has descendants.
- o $EXTRACT - Extracts a fixed length subfield from a composite string.
- o $FIND - Determines at which position a substring occurs within another string.
- o $JUSTIFY - Performs right justification on a string.
- o $LENGTH - Returns the number of characters in a string.
- o $NEXT - Returns the next subscript at the same level of the GLOBAL.
- o $ORDER - Places non-integer subscripts in logical order.
- o $PIECE - Separates delimited substrings (known as PIECEs) from a larger string.
- o $RANDOM - Returns a pseudo random number.
- o $SELECT - A CASE statement of sorts that evaluates a list of expressions and returns a value associated with the first true one.
- o $TEXT - Returns a command line of MUMPS code from a specified location.
- o $VIEW - Allows the user to examine machine dependant information specified in the implementation.

CONCLUSIONS and OBSERVATIONS

MUMPS has a public relations problem. It has never gotten the attention it deserves, considering its advantages and installed base. It was so progressive and advanced when it was developed in the mid 60's that its advantages were not understood or appreciated. Its concentration in the medical field has not helped, nor has its concentration in the VAR and OEM market. This could be a disadvantage in attracting top technical talent, justifying purchases and other areas.

MUMPS has historically been an environment of its own with no need for outside utilities or operating systems. This has been a great advantage in the integration area, but it has also limited MUMPS' connections to the outside world. There are proposals before the MUMPS Development Committee to provide standard ways to access data from, and provide data to, other environments.
In my opinion it has also made the MUMPS community ingrown and less aware of outside trends and developments in the world of computing. This, along with the ability MUMPS gives the programmer to abbreviate all commands to one letter leads to a culture that encourages a very terse and difficult to read programming style, among other things. This stems in part from the early MUMPS implementations that only allowed a small amount of code in a program, causing abbreviation and stacked up commands to the extreme. Although these limitations are gone in recent implementations and the most recent standards provide all the tools necessary to write well structured code, the MUMPS community has been slow to see the need to change and to respond. All recent thought in the data processing community has been that you spend many hours supporting a program for each hour you spend developing it. Development shortcuts can cause you to hemorrhage money later as you try to support un-readable code. I have heard MUMPS programmers say that it is easier to re-write the program (which is easy and fast in MUMPS) than to figure out what it is doing (which is hard, the way a lot of MUMPS code is written). This causes a whole new set of bugs. I have been told by top technical personnel in more than one company that they can read code easily so there is no problem. The test should be "can anyone that has programmed for a couple of years but does not know MUMPS pick up a MUMPS program and read it pretty well". That would not be as near possible with the MUMPS programming style I have observed as it is with most other languages. As hardware speed increases at an ever increasing rate and software becomes more of a limitation, the ability to support your software in a productive manner will be at least as important as the ability to develop it productively. This is a problem with MUMPS culture, not the language and it could be corrected at any particular site with visionary leadership.

It is apparent that MUMPS was way ahead of its time when it was developed about 20 years ago. Its key advantages, interpretive prototyping, portability, dynamic transaction oriented database, imbedded database language, are the most trendy and topical issues of today. Since the rest of the computing world now understands these issues as well as the MUMPS developers did then, other commercial ventures offer languages with a similar list of advantages. How then can MUMPS distinguish itself from the likes of Oracle, Sybase, Powerhouse, Ingres and the host of others out there? It would be outside the scope of this paper to do a point by point comparison, and I have a feeling that each language would come out with list of winner and loser issues. One point that makes MUMPS stand out is that it is governed and defined by a non commercial body. Most of the other similar languages that I am aware of are proprietary languages sold by a commercial entity. Entrusting your applications to the most benevolent commercial interest (dare I say even Hewlett-Packard) can put you in a position that ranges from uncomfortable to very expensive as their priorities diverge from yours. Many members of the audience would have stories to tell about broken promises, "vaporwear", and misrepresentation in order to make a sale. At least with the Mumps Users Group you can influence the direction of the language before the fact. Since the MUMPS language, and a lot of software written with it are in the public domain it tends to be a low cost solution.

The MUMPS Development Committee is working on proposals in the following areas:

- Proposals to standardize parameter passing, communications with other languages and calling external procedures.
- Distributed databases, networking, interoperability.
- Logic programming capabilities.
- Object oriented processing capabilities.
- MUMPS technology bindings to other standards.
- Security issues in the open MUMPS environment
- Development of a standard MUMPS library specification.

- Richard F. Walters, Is MUMPS the Cure?, Microcomputing (Feb. 1982)

- Richard F. Walters, J. Bowie and J. C. Wilcox, MUMPS PRIMER, (MUMPS Users Group, 1982)

- Philip Blume, The Impact of MUMPS on System Development, Software in Healthcare (February/March 1985)

- Walter R. Houser, Software Engineering, Programmer Productivity and Systems Performance, Conference Proceedings of the 1984 International Conference in Computer Capacity Management, (1984)

- Paul Grabscheid, The "Relational-ization" of MUMPS, Computers in Healthcare, (MUMPS Special Edition, 1988)

- MUMPS NEWS, Volume 6, Number. 4, 1989

- Steve Koerper, MUMPS Starter Kit, (Hewlett-Packard)

- Martin Johnson, Abstract for a brief recitation of the history of the MUMPS language and its supporting organizations.

- Eric Skjei, The Mind Behind MUMPS, Computers in Healthcare, (May 1989)

Real-Time Control of a Metal-Organic Chemical Vapor Deposition Facility

Wayne Gerdes
NASA Langley Research Center
MS 473
Hampton, VA 23665-5225
(804) 864-1520

Abstract

The design and implementation of an interrupt-driven, real-time, distributed, fail-safe control system is discussed. This system uses Pascal software to control a Metal-Organic Chemical Vapor Deposition (MOCVD) system. The control system consists of a Hewlett Packard (HP) 9000 Series, Model 220, computer communicating with an HP 2250 Measurement and Control System via IEEE-488 [also known as the HP Instrumentation Bus (HP-IB)]. The HP 220/2250 combination controls a MOCVD facility used for research into the processes involved in the growth of semiconductor structures with compositional-layer superlattices. The critical elements of this control system were implemented using double buffering, interrupt-driven data downloads, watchdog timers, and distributed control. The control system successfully responds to an interrupt from the HP 2250 and downloads data within a critical design parameter of 100 milliseconds. Along with the software design discussion, comments are given concerning the problems and avenues for possible improvements noted during the project.

Introduction

Metal-Organic Chemical Vapor Deposition (MOCVD) systems require the safe, real-time, flexible, reliable control of a variety of toxic and/or flammable materials. To meet these demands, an interrupt-driven, distributed control system concept, with a maximum cycle time of less than 100 milliseconds, was selected. The requirement for a 100 millisecond response time comes from a need to accurately control the growth of multiple, very thin layers of semiconductor alloys for research into the use of superlattice structures for advanced electro-optic devices. The semiconductor research requires the growth of 100 to 200 layers of different alloys with each layer thickness independently chosen over the range of 2.0 to 20.0 nanometers. The switching to achieve these growths is performed in the plumbing manifold leading to the MOCVD chemical reaction chamber. The control system to direct the MOCVD system in the production of superlattices employs an HP 9000 Series, Model 220, (Master) for master control and an HP 2250 Measurement and Control System (Slave) for Input/Output (I/O). The control software was written in Pascal and Machine Control Language (MCL/50). The goals of the project are discussed along with the history behind the selection of Pascal for implementation. Implementation of the real-time, interrupt-driven software is presented in detail. Suggestions for avoiding obsolescence in future systems and notes on the problems encountered during this project are given.

Design History

A brief explanation of the terms "BASIC Operating System" and "Pascal Operating System" may prevent confusion for those who think of BASIC and Pascal as languages. The BASIC Operating System is a form of HP's Rocky Mountain BASIC and performs all operations with BASIC program statements. This is in contrast to the more traditional Pascal Operating System, where operations are performed by a variety of separate utilities. This separation of duties allows the Pascal system greater flexibility at the cost of making some operations difficult to perform within a program. The Pascal system and its utilities are written almost entirely in $MODCAL$, an HP experimental version of Pascal.

The control software was initially written using the HP BASIC Operating System for a number of reasons. First, the BASIC Operating System contains an interrupt construct. Second, the only HP 2250 control software examples available from HP for desktop computer control were in BASIC. Third, Rocky Mountain Basic lends itself to writing instrument control software. Problems were encountered initially in establishing communication between Master and Slave. Differing End-Of-Line (EOL) protocol between these systems, the HP 2250 with FORTRAN EOL and the 9000/220 with several different possible BASIC EOLs, was one of the problems encountered. The BASIC command that solved this particular difficulty was: 'ASSIGN @SLAVE TO 806; EOL "line-feed-character" END'. The single quote marks and period are not part of the syntax for the command and line-feed-character refers to the single ASCII character. The use of secondary addresses on the IEEE-488 (also known as HP-Interface Bus or HP-IB) by Slave was also a source of complications. Use of the BASIC operating system was abandoned when difficulties were encountered in transferring data to secondary addresses in the HP 2250 using BASIC at interrupt priority. The BASIC system would not change the data in the HP 2250 until the second attempt. When HP was unable to reproduce this problem, it became necessary to rewrite the program in a different language. The HP Pascal system was chosen for the second effort.

Rewriting the control software in Pascal was simplified by using communication subroutines written for NASA by John Perry and C. W. Stroud.[1] Use of their I/O routines solved the secondary address communication problems under Pascal. The next challenge involved implementing interrupts. The manual set distributed with the Pascal system contains no information about connecting to IEEE-488 interrupts. The information and software required to use these interrupts and much other useful information, is contained in the Pascal System Designer's Guide. Most of the time critical and device interface code contained in the HP 9000/220's control software would have been impossible to write without this documentation.

Equipment

The MOCVD system consists of four major subsystems: the computer control system (fig. 1), the control panel (fig. 2), the MOCVD reactant supply manifold and growth chamber (fig. 3), and the exhaust systems. Figure 4 is a symbol table for figures 1 to 3.

As previously mentioned, the computer control system consists of an HP 9000 Series, Model 220, (Master) and an HP 2250 measurement and control system (Slave). Master is a Motorolla 68000 based system equipped with floppy and hard drives communicating via an IEEE-488 bus. Slave, based on an HP 1000 processor, is connected via a separate IEEE-488 bus to Master. Slave is responsible for all I/O to the MOCVD system. It provides closed loop control of the hardware via sensors monitoring critical parameters. Feedback for closed loop control of the relay-actuated, solenoid-triggered pneumatic valves is provided by microswitches that verify their operation. Closed loop control of the mass flow controllers is provided by analog inputs and outputs. Thermocouple inputs are used to monitor various MOCVD temperatures. Digital inputs monitor power fail, manual stop, flammable gas alarms, and exhaust system alarms. Of special note is the Slave watchdog timer command that is used to detect computer lockup. This command is used to perform an emergency shutdown of the MOCVD system if not reset every 100 milliseconds.

From a control viewpoint, the critical features of an MOCVD system are the toxic and/or flammable nature of the materials involved and the special switching considerations. The time constraint is a result of the abrupt switching between different organometallic sources that are necessary to create superlattices. The system must produce a sharp, clean transition between the gases used for growing one layer and those for the next layer. The 100 millisecond value is derived from a combination of the growth rates and desired layer thicknesses for this system. Special measures have been taken to reduce stagnant regions, tubing length, and residence time in the MOCVD reactant supply manifold and growth chamber.[2,3]

Software Design

The overriding design criterion, after safety considerations, is the necessity of performing all operations within 0.10 second (100 milliseconds). Within the allowed time, the Slave control system must be able to properly configure all the valves, mass flow controllers, and other hardware. This time must also include occasionally downloading new control data to Slave. The data download is required because Slave has insufficient memory to contain a control program that may run for days.

Some of the examples of the software design decisions that arise out of the 100 millisecond time constraint are: double buffering data read from Master's disk drives for downloading into Slave, the use of interrupts to signal for these data downloads, and the multiprocessor control system design. Double buffering allows the system to have one buffer ready to download while loading the other. The amount of data that can be transferred to Slave is governed by the worst case response time of all controlled hardware. After Slave issues commands to the device controllers, it must wait for the relatively slow mechanical devices to function. The time between issuing commands and checking for feedback is available to download command data or for secondary processes such as the temperature measuring subroutine. The amount of time available is discussed below.

The process of asking Master for data and getting it stored in Slave was measured using a Tektronics DAS 9200 Digital Analysis System at the 200 nanosecond clock setting. A representative sized data transfer of 100 integers was selected for timing purposes. The time required to perform the complete operation was 14.1 milliseconds. This was measured from the time the IEEE-488 SRQ (Service Request) line went true to the last transition of the Not Data Accepted (NDAC) line on the transfer of last byte of data. Data was clocked into the analyzer by monitoring NDAC. The time history of the transfer is presented in figure 5. A more complete description of the data transferred is given in the paragraph describing start-up of Slave. Master requires less than 1.4 milliseconds to respond to the interrupt and only 2.2 milliseconds to transfer the actual data. Various overhead operations require approximately 10.5 milliseconds. Once finished with the overhead and shorter information downloads, the transfer of the bulk of the data, 200 bytes, takes place at more that 1 megabaud.

Using interrupts allows Master to loop through a MOCVD status display while waiting for Slave data download requests. The alternative would require Master to continuously check status buffers in Slave. While performing these checks (polling), Master would be unavailable for useful work. Slave requests data via setting the IEEE-488 SRQ line.

Using information and IEEE-488 service routines from the Designer's Guide,[4] it was arranged for an interrupt to occur in Master on receipt of the SRQ. This was accomplished using module "hpib_5_init", procedure "on_srq", and the example program from the Designer's Guide.[5] The rate at which data could be loaded for transfer was significantly improved by the use of special buffers.[6]

When the SRQ from Slave forces an interrupt, it is critical that the interrupt service routine (ISR) be kept as short as possible. This is because the IEEE-488 routines interrupt at priority three, which is a higher priority than most of the rest of the Pascal system. As a result, lower priority operating system routines, such as those that service the system clock, cannot execute until this routine exits. The ISR is about 50 lines long including subroutines. For some reason, only the "fast" programmed loop HP Pascal I/O transfer statement will work in the ISR. Use of the direct memory access form of the transfer statement would improve the transfer rate. An improvement in the response time might be gained by rewriting these routines in assembler. This approach has not been taken because most of the code executed at interrupt priority consists of calls to the IEEE-488 bus drivers. Gaining any significant speed improvement would require rewriting these drivers in assembly language. The description of the software design parameters is now complete. Next is a detailed description of the software.

The following describes the initialization sequence between Master and Slave. Master is started first then Slave. When the MOCVD control program is started, a variety of initialization routines are executed. These routines perform the following functions: filling the data download buffers, setting time-outs on the IEEE-488 transfers, and clearing the display terminal. A number of routines are downloaded into Slave at this time. In order they are: three routines that initialize Slave, a crash stop routine, and finally the control program. As the programs are loaded into Slave, they are automatically compiled and run.

The primary data structures in Slave are:
  B1, the command queue, contains the actual data;
  V1, points to the data used for the current time step; and
  V2, points to the last data item (end of data).
The data for each time step consists of 12, 16-bit words. The first word is an integer containing the number of 100 millisecond steps to delay before continuing execution (sleep timer). The next three words are bit encoded valve setting instructions, each bit representing a valve state, one for on, zero for off. The last eight words are used to generate analog voltages to control the mass flow controllers.

When the Slave control program starts, it immediately finds queue B1 empty and requests data by setting SRQ. On receipt of the SRQ, Master jumps to the ISR at priority three. The first action of the ISR is to conduct a SRQ serial poll to determine what device is requesting service. Next the ISR reads 16 integers from Slave's secondary address 4. The 16 lowest HP 2250 internal interrupts, which are separate from HP 9000/220 interrupts, are kept in this buffer. HP 2250 protocol requires that these integers be read before another SRQ can be generated. Since they must be read, they are used to return Slave's status. If the first of these values is a one, the data transfer is started. The other standard return code is "finished", value 99. If any other value is returned, a crash stop of the MOCVD system is executed by both Master and Slave. The actual data transferred when Master services the interrupt consists of two transfers. The first updates variables V1 and V2 via secondary address 7 with the following 16-bit integer information: the starting variable number, the number of variables, and the new contents of the variables. The second transfer, via secondary address five, updates command queue B1 with the following information: the destination buffer number, the number of bytes to be transferred, and finally the new buffer contents. This concludes the distributed control system initialization. After startup, Master loops through status display routines until a data request or shutdown command is received. Slave now controls the MOCVD system.

When a Slave program time step starts, variable V1, the command queue buffer pointer, is incremented by 16, so that it points to the next set of commands. The first data word in this set of commands is the sleep timer which controls the number of times Slave will execute a wait loop. This controls the number of 100 millisecond time periods that the last set of commands will remain in effect. This number may be zero or more time steps. This timer also has the special value -9999, used as the end of run flag. After the wait loop concludes, the timer is cleared and set one more time for a 100 millisecond delay. Variables V1 and V2 are then tested. If they are equal, Slave interrupts Master via setting the SRQ line, either to request more data or signal end of run depending on the end of run flag. If there was a request for data, V1 is reset and V2 is set to the new end of data during the download.

After initializing the various pointers, the "slow" (40 millisecond[2]) mechanical valves and mass flow controllers are configured. The program then waits for the valves to function before checking feedback from the various devices. Since the actual time required to set and check the various pieces of hardware is on the order of 10 milliseconds, approximately 90 milliseconds will normally be available for other tasks. This time is used for low priority Slave programs such as the one that monitors temperatures. To manage just this kind of problem, the HP 2250 has a hierarchical multi-tasking operating environment. Once a high priority task gives up control, the next lower priority task runs to completion. As the hardware feedback must be tested approximately 50 milliseconds after the hardware is set, the lower priority programs must perforce execute in much less time, to prevent minor timing variations from causing a crash stop. If the lower priority programs fail to complete in the required time, the watchdog timer will not be reset and a crash stop will occur. The time slice is now complete and both computers continue in the same manner until normal termination or an unexpected event occurs.

In addition to the implied run command discussed above, two other commands currently exist. Master can command Slave to crash stop the growth process. This can be the result of an operator input to Master or external events such as a power failure continuing more than 5 minutes. The default crash stop routine duplicates a manual emergency stop by shutting all valves, or a special crash stop routine can be loaded at start-up. The other command is a slow step command used to validate a control sequence. The system slowly steps through the growth commands so they can be checked for correctness.

Special Considerations

When using the HP 2250 as part of a distributed control system, care must be taken to avoid disabling communication between the HP 2250 and the IEEE-488 bus master. The HP 2250 interprets any extraneous input as the start of a new main task. As part of preparing to execute this bogus task, the HP 2250 dedicates its communication hardware to returning the contents of the main task buffer. Until the main task buffer is read, the HP 2250 communication hardware is unavailable for other tasks.[7]

When dealing with interrupts, a number of problems not normally faced by high-level language programmers come up. Most of these are covered in the various manuals distributed with the Pascal Operating System or the Designer's Guide. One that is not explicitly covered is the problem of creating non-interruptible code. Certain code sections, formally called critical sections, must not be interrupted. The critical code in this program is in the routine that fills the download buffers. It would eventually "crash" the Slave control program if the interrupt routine grabbed data from a partially filled buffer. One common method of dealing with this problem is to prevent interrupts during the critical section. Since the disk drives and the control program both use the IEEE-488 bus interrupt, this method was not available.

The control program deals with the problem by setting a "do-not-disturb" variable or more formally a semaphore. The ISR checks this semaphore and, if true, increments an "interrupt occurred" variable and exits, returning control to the critical section. The critical section checks the "interrupt occurred" variable on exit and calls a special subroutine if it is greater than zero. This special routine is called a fake interrupt service routine (FISR) in HP terminology. The FISR first performs the normal ISR functions then decrements the "interrupt occurred" variable. In this program, only one FISR is allowed before a crash stop is initiated, because executing a FISR implies that one data buffer is empty and the other partially filled.

The urgency of porting the distributed control software to more modern hardware is underlined by the official retirement of both the HP 2250 and the HP 9000/220 by HP. However, when the design of this software was started, there were no real-time system call standards to which the code could have been written. As a result, the code will be difficult to port to newer, faster computer hardware. A partial solution to the problem would be the use of "open system" specifications like POSIX IEEE P1003.1 Portable Operating System Interfaces with the P1003.4 real-time UNIX extensions. The use of these or similar specifications may not entirely solve the problem. The P1003.4 standard is still in committee and, as of this writing, leaves many details to the software implementors. An example of detail variations among the implementations that could cause portability problems is timer resolution. One implementation might support 100 nanosecond timer resolution and a different implementation support 1,000 nanosecond resolution and both would be within the standard. ADA has also been suggested to help eliminate portability problems. However, the various implementations of ADA can also differ enough in the details of their real-time support to make transportability difficult.

The design of this system was complicated by the lack of multi-tasking constructs in the HP Pascal system. It is desirable to design real-time control systems as multiple processes cooperating on separate processors to perform the required tasks. It is very desirable to write even a modestly complicated control program via hierarchies, priorities and interrupts. Low priority is used for the things that do not change "quickly" such as the measurement of temperature and the user interface. With this class of routines, timing uncertainties of a few tens of milliseconds are not significant. Other types of routines will execute only seldom but must preempt everything else. Examples of such routines include those that deal with fire sensors, power fail, and emergency shutdown situations.

In dealing with real-time control software, error recovery becomes an important topic. Through the use of supervisor mode error trapping and recovery tools, user program crashes and hang-ups can be detected and corrective action taken. Often the only corrective action that can be taken is an emergency stop, but this is preferable to uncontrolled operation of a potentially hazardous system. The Pascal system has a "Try...Recover" mechanism that does operate in supervisor mode but this does not cover all cases. Currently, the Pascal system returns the message "Restart with debugger?" when a user mode program crashes. It would be useful, in this case, to substitute a crash routine that would attempt to clear the IEEE-488 bus and give recovery commands to Slave. The advantage of a supervisor mode operating system is that this type of recovery operation is possible, as opposed to simply locking-up, as computer systems without this feature are prone to do.

Conclusions

Through the use of a variety of real-time techniques, primarily interrupts, the Pascal Operating System can be used as part of a critical process control system. Programs written under the Pascal system can respond to interrupts in less than 1.4 milliseconds. Transfers of significant amounts of data over the IEEE-488 bus, even at interrupt priority 3, may take over 14 milliseconds. The version of Pascal supplied with the Pascal Operating System has a large number of enhancements over "standard" Pascal that make it much more suitable for real-time programing. This enhanced version of Pascal, in conjunction with the System Designer's Guide software, is adequate to the task of writing efficient, reliable real-time software. Some knowledge of system driver software is helpful when programing software this tightly integrated with the hardware. Since the Pascal system was not developed specifically for real-time control work, there are a number of areas that require user ingenuity. These areas include communication with secondary addresses on the IEEE-488 and error recovery tools that run in supervisor mode. These difficulties were overcome and the required software developed and tested.

The multi-tasking environment of the HP 2250 allows efficient real-time control programs to be developed. However, care must be taken to provide exactly the correct inputs at all times. The flexibility of this device would be improved if it supported variables in more commands. Despite these restrictions, the HP 2250 has been sufficient for this demanding control task.

Retirement of both the HP 2250 and the HP 9000/220, emphasizes the necessity of writing transportable programs. The use of standards, such as the IEEE P1003 series, is suggested to reduce the probability of software obsolescence and to prevent dependency on any particular piece of hardware.

It is hoped that the discussion of the safety features of this system may be of use in the design and operation of similar facilities. The use of safety related concepts such as supervisor mode error trapping and recovery software, watchdog timers, and distributed control should be considered in any similar system.

[1] Perry, John and Stroud, C. W.: "Operation Of The HP2250 With The HP9836A Using Pascal 2.1," Jan 12, 1984.

[2] Clark, I.: "MOCVD Requirements For Abrupt Junction Growth," Phd dissertation, UVA, 1990.

[3] Clark, I.; Fripp, A.; and Jesser, W.: "MOCVD Manifold Switching Effects On Growth and Characterization," Proceedings of the 8th American Conference on Crystal Growth, Journal of Crystal Growth Special Issue.

[4] Pascal 3.0/3.1 System Designer's Guide for HP 9000 Series 200/300 Computers, Order No. 98615-90075, Dec. 1985 Edition 1, pg. 11-2.

[5] Pascal 3.0/3.1 System Designer's Guide for HP 9000 Series 200/300 Computers, Order No. 98615-90075, Dec. 1985 Edition 1, pp. 538-543.

[6] Pascal 3.0/3.1 System Designer's Guide for HP 9000 Series 200/300 Computers, Order No. 98615-90075, Dec. 1985 Edition 1, pg. 512.

[7] HP 2250 Measurement and Control Processor Programmer's Manual Update 1, Sept. 1982, pg. 11-2

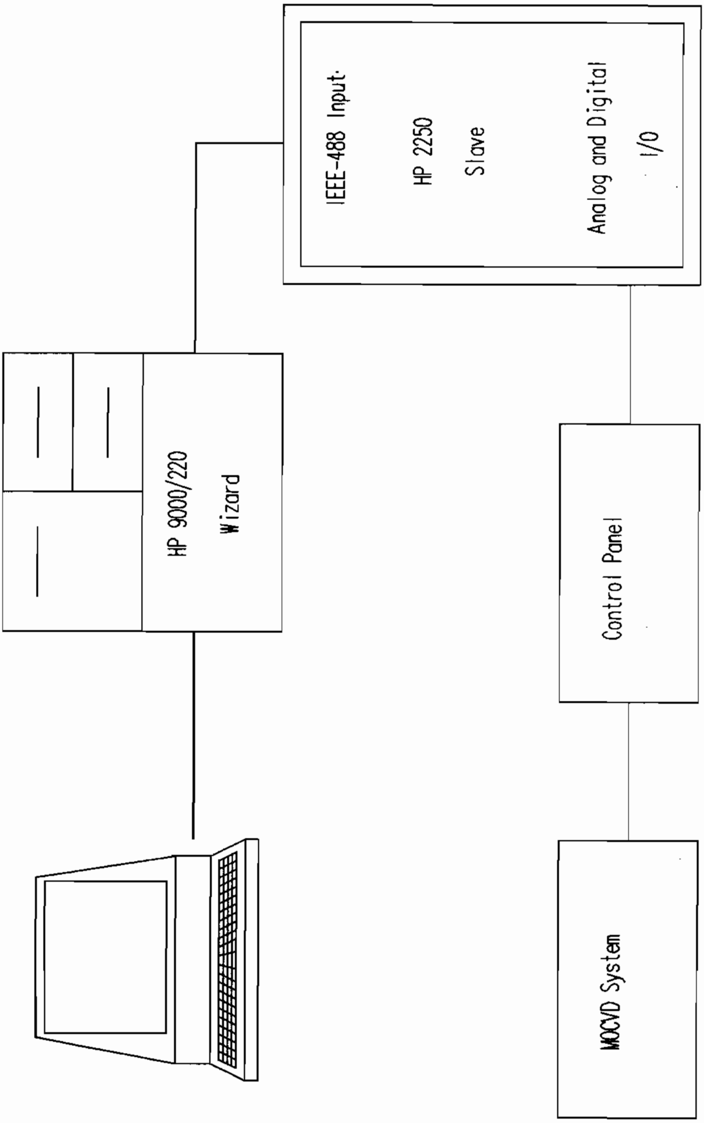


Figure 1
MOCVD Computer Control System
Real-Time Control Of A Metal-Organic Chemical Vapor
Deposition Facility
1008-11

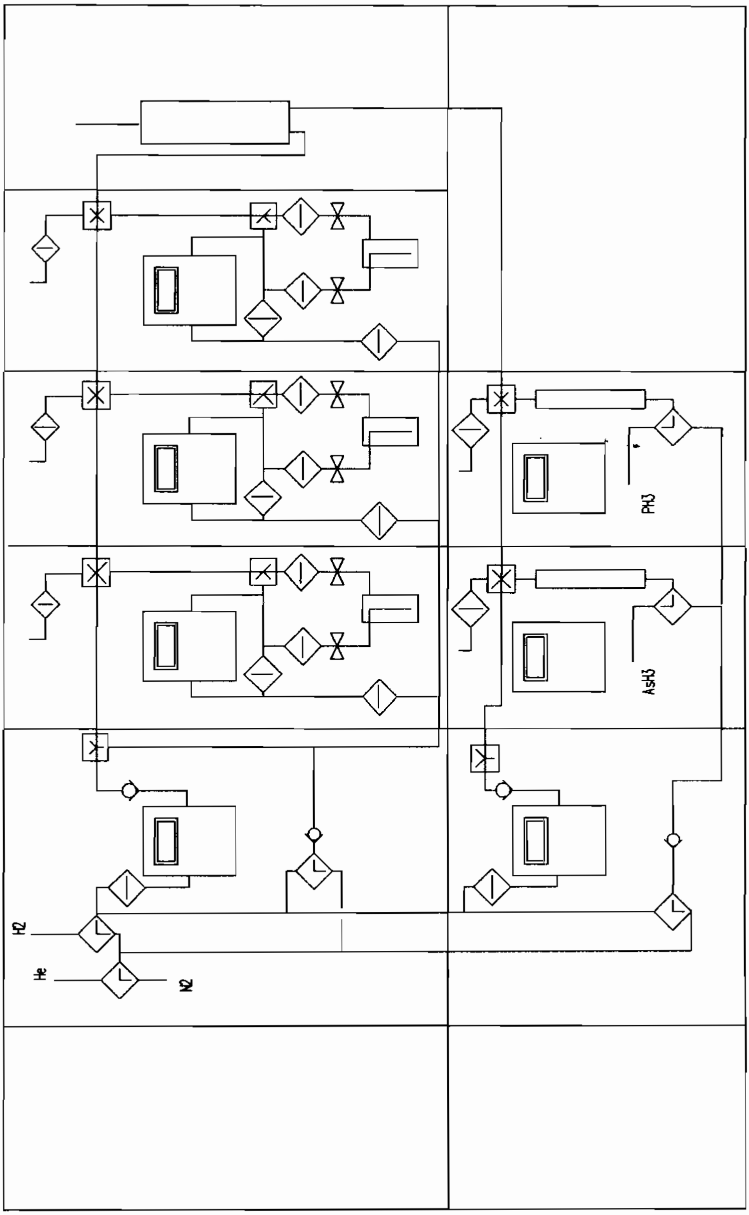


Figure 2
Real-Time Control Of A Metal-Organic Chemical Vapor
Deposition Facility
1008- 12

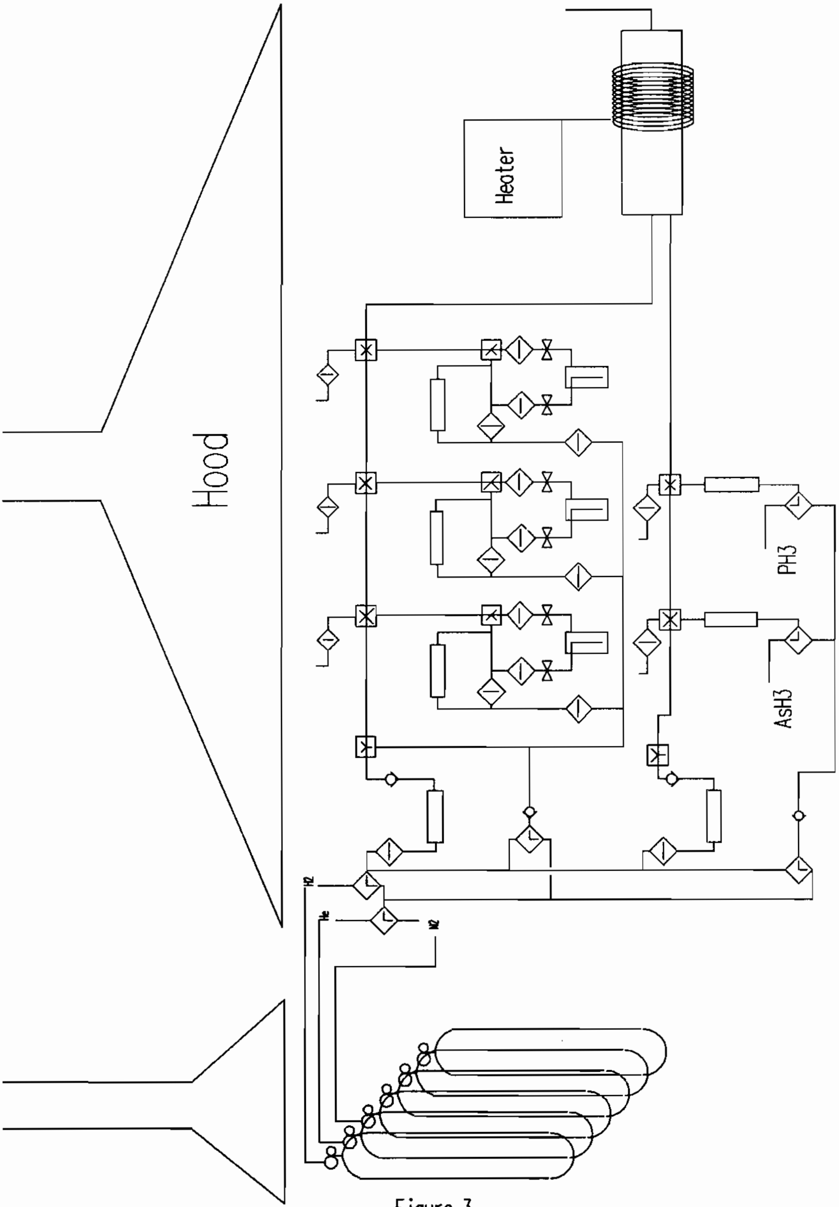


Figure 3
MOCVD Reactant Suppy Manifold And Growth Chamber
Real-Time control Of A Metal-Organic Chemical Vapor
Deposition Facility
1008-13

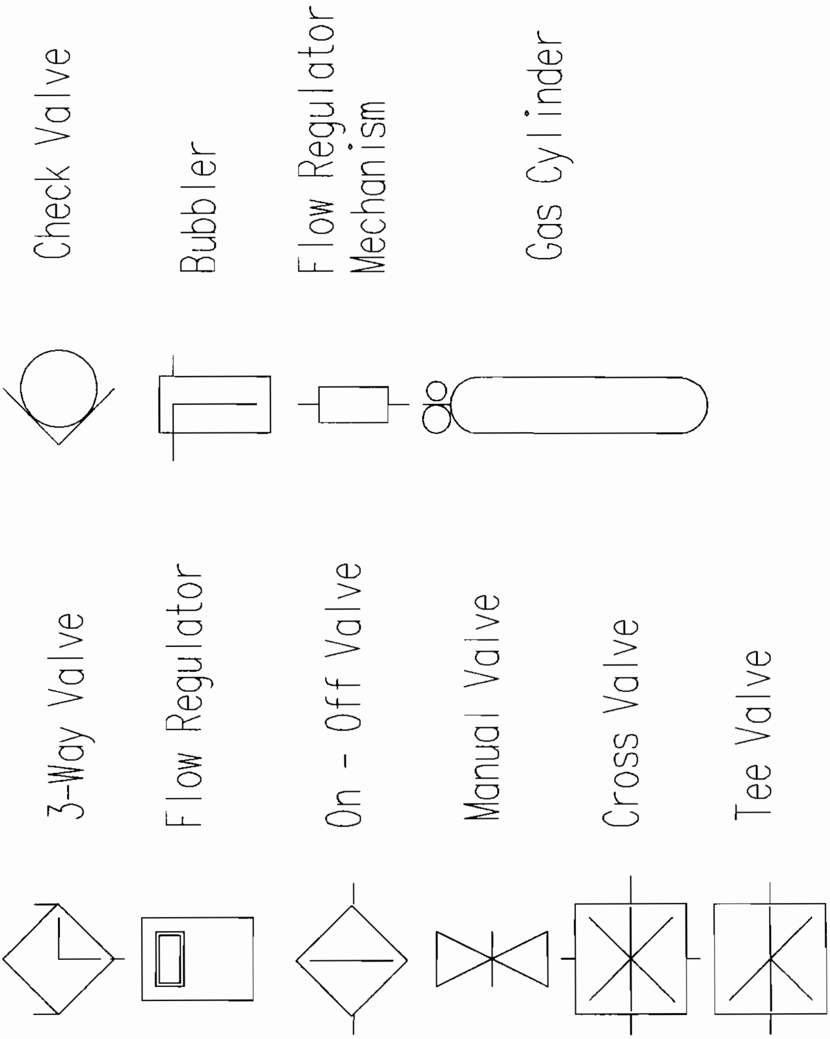


Figure 4
Symbol Table
Real-Time Control Of A Metal-Organic Chemical Vapor
Deposition Facility
1008-14

```
--- Time Zero - SRQ Start

--- 1.4 milliseconds - Master responds to SRQ

--- 9.2 milliseconds - Start of 16 HP 2250 Integers
--- 9.5 milliseconds - End   of 16 HP 2250 Integers

--- 11.9 milliseconds - Transfer New Variables
--- 12.1 milliseconds - Transfer Buffer No. & No. of Data
--- 12.5 milliseconds - Start of 100 Integer Transfer

--- 14.1 milliseconds - Transfer Ends
```

Figure 5
Interupt Driven Download Time History

Real-Time Control of a Metal-Organic Chemical Vapor
Deposition Facility
1008-15

Real-Time Data Collection in a Manufacturing Job Shop

Steven L. Blickensderfer
3M Company
610 North County Road 19
Aberdeen, South Dakota 57401
Telephone 605-229-5002

Introduction.

Data collection on the factory floor is an activity that is changing rapidly. Many of the changes that have taken place in the office have moved out into the production areas. Just going back ten years gives a perspective on how the number of tools and the equipment available has changed. The use of automatic identification methods like bar code I.D. has exploded. Dozens of terminals and devices have been introduced that are built to withstand the rigors of the shop floor - and still be affordable. This rapid change is even more bewildering to those who must make decisions on factory systems when you consider that a major project can take two years to plan, build, and qualify, sometimes longer.

How to enhance or ease the cost of your facility's floor data collection needs is not an easy question. Data collection varies so much between operations. Perhaps more difficult to conform to are accounting cultures and practices. Vendor solutions are available, but even with the floor interfaces supplied, it is still a major investment to implement a solution that satisfies everybody involved. It seems that there is always something missing.

About eight years ago, the 3M Aberdeen tape slitting department was faced with a dilemma. We were investing a lot of time and money in a production reporting system that was out-dated and ineffective. The direct savings were there to implement some sort of an automated reporting system. The problem was how to go about it. This paper will describe the operation, the reporting problems that were encountered, and the solutions that were used, including our choice of the HP1000 with in-house programming as the main platform for the system.

A textbook "job shop". 3M Aberdeen Tape Slitting.

Our tape slitting department is an operation that has grown from a 3M ownership of adhesive technologies and a corporate culture of capitalizing on opportunities that arise from solving our customers' problems. The department converts about 120 different kinds of adhesive tapes from "jumbo" rolls supplied from different 3M locations. The tapes are cut by a bank of circular knives and alternately wound on two output mandrels. The tapes are produced in any reasonable width and length.

There were several reasons why the Aberdeen operation was producing such a crushing load of paper-work:

Low volume orders.

The bulk of our customers are industrial customers who use our product in their own processes. Although a few of our products are stocked in finished form, most are shipped directly from our plant within 6 days of receiving the order. Many orders take less than an hour to produce and pack out; some of our production cells average 15 or 20 orders per day. Some scheduling is possible to reduce change-over time, but is common for each order to require a complete change of input rolls, labeling, and packaging materials.

High value products.

The tapes that are produced in Aberdeen are specialized and expensive to produce. Any gains in yield that can be purchased through more detailed production data are pursued. Waste-by-cause and total yield data is collected and closely watched. Most of the products have between 20 and 30 reasons that waste can be attributed to, and there is a high level of rework to recover tape from rejected material.

Another area affected is Quality Assurance. All of the products are subject to our customers various quality standards and testing demands. Lot traceability and quality test data are kept for all of our products, since most of them are constantly undergoing change or experimentation.

Involvement of different operating units.

Several 3M divisions do converting work at Aberdeen. The resources used are charged off directly through the production reports. Since many of the products are experimental, this data is useful in determining a products profitability, especially if it has properties that make it difficult to cut or hard to wind.

Although these reasons are responsible for most of the data collection we do, there were other problems that were caused by the lack of accuracy and discipline in our reporting practices. One had to do with individualized incentive that was awarded according to work content reported. There was a tendency to stretch the truth on many items that could not be checked, and this hurt the credibility of the incentive program, along with the direct cost of the payments made on embellished information. Another problem was maintaining inventories on our input stocks. Many operators would work "backwards" on their production reports, calculating the amount of input that they used by figuring how much they should have needed to produce their output rolls. As a result, our inventories were constantly needing to be monitored and adjusted. In addition to the direct costs for data collection, there were other considerations. When you are dealing with a job shop operation, there are so many products involved, and they are produced on an intermittent basis. The accounting data collected on the floor is an important part of the business, because it must be examined continuously to ensure that the profitability of the products is being maintained. Going to an accounting formula to ease labor hour reporting, for instance, could ease our reporting costs - but decision support would suffer.

It was clear that there was an opportunity for building an automated reporting system. Our reporting load was not going to be reduced. The higher volume products were being pulled out and slit in other locations, and we were getting newer, smaller lot materials to produce. Time studies calculated the time spent reporting at about 11% of our total production time. Process engineers were sifting through written reports to find answers for high waste or low rates. The factory accounting effort to process the reporting from the production area was normally 6 to 8 people - for only 10 machines! An automated reporting system would pay off, but it would take a large investment in time and money.

Challenging your reporting practices - "liberate", then automate.

When you take on the task of reducing the data collection costs in your operation, the best place to start is by challenging your current reporting and accounting practices.

Computer terminals and bar code devices are becoming common on the factory floor, but the bulk of the factory floor accounting today is done the old-fashioned way, with paper and pencil. Usually there is some type of form used that the controllers have designed, with columns for the inputs and outputs. One thing that is common with many of these forms is that they will be designed to be taken directly to a keypunch step, with all of the information laid out according to your accounting system. Many times a savings can be had just by rearranging the form in a more efficient manner, then using a front end program to build the input for the accounting system from data input from the modified form.

Reducing the amount of detail in your production reporting should be considered before beginning automation. An analysis of the data on each production report may reveal that some of the input could be obtained from other sources, and other inputs may be found to have little value in the decision support process. Although accountants are always reluctant to use a "formula" charge off for some items, many times it just makes good business sense.

The Aberdeen operation had a good example of a reporting activity that had little value. Before the advent of automated reporting, we always reported the packaging items on each production report. With the wide variety of products, there was a lot of effort to do this; keeping the correct items on each report, reporting substitutions and throwaways, and key-punching the data. The usages of the packaging items were never correct, and large adjustments were common after counts. What was brought to light was that packaging costs, on the average, were only about 5-8 percent of our product cost.

The reporting was being done this way so that the cost of commonly used items could be distributed correctly to different products. Analyzing the products produced over a several month period and the packaging used by each gave us a formula to distribute the packaging costs fairly. Since our packaging costs were so low as a share of total unit cost, we were able to sell the formula to the accountants. The packaging was removed from the reports, and all of our problems disappeared. Certainly, there are instances where the formula will fail, or is inaccurate, but the cost savings from reduced reporting makes up for any inconvenience - and it is no worse than what we were doing before.

Another way of reducing reporting is to go to a formula basis for items that are related by bill of materials, and this is commonly done at 3M. The operator reports only their output, and the inputs are calculated and filled in by the accounting system according to standard usages. This will be troublesome if you incur waste or will frequently substitute items; an altemative that works better is to examine records for the inputs delivered to the department and distribute what was delivered to what was produced. If you can achieve workable and accurate cut-off points, this arrangement seems to work well.

A high amount of detail in production reporting is always difficult to achieve for a job shop operation. In effect, you are placing responsibility for your accounting in the hands of the people who write reports. Although there are practices that can reduce the reporting cost, the reporting accuracy still depends on the discipline and timeliness of each individuals' own reporting habits. This can lead to a never ending cycle of training, inventory adjustments, and report auditing. This is why the Aberdeen group chose to go the extra step and install real time features in our reporting system.

Real time - what is it, and is it worth it?

Real time data collection means different things to different people. Many think that the presence of a terminal on the factory floor is real time reporting. Perhaps.

Real time is collecting data about production events at the time that they happen. Where it gets interesting is when you begin to debate in what detail or how narrow the time frame between reported events. It may satisfy all of your needs to have the operator input information into the terminal every hour, or shift, or at the end of each order filled. If your objective is to improve your reporting accuracy and enforce some type of discipline and timeliness, you may need to go further.

If there is a continuous reporting activity that goes on in your area, you can build an effective system by using some real time process control. Build the programming around break points, and at the break points, have some sort of reconciliation between the inputs and outputs. The other key to real time is to use some form of data capture, because real time is more effective with more frequent input. Where you must put your thought is in having enough interaction to make the system effective without interfering in the process.

With some very simple edits, done frequently, we were able to build an effective system that, in addition to increasing reporting accuracy, enforced good working habits and timely reporting. The tape operation has two primary break points. One is at the end of a "cut", or the point at which one output mandrel of tape has been wound to the required length. At this point, the counter stops the machine, and cannot be reset to zero except by the HP1000 computer. The operator must initiate a touch on the screen to signal the end of cut, and a reconciliation of the inputs against the outputs is done. The program obtains the count from the encoder on the input roll and the count from the output mandrel, and calculates to see if the inputs and outputs are balanced. For instance, if there was a waste occurrence of five yards of input web that went unreported, the input counter would have five extra yards. The operator is notified, and the error is corrected before the counter can be zeroed and the process allowed to continue.

The second break point is at the end of the report. The system has been counting the rolls as each cut is made; the cuts times the rolls produced by each cut should equal the output - minus any rolls that were discarded or sent to rework. We ask our operators to keep their totals up to date. When the end of the report is reached, a quick count of the good output rolls allows for a reconciliation and a chance to correct any errors before making the final entry. If the operator is delaying the reporting of the waste rolls until the end of the report, it is easy to catch.

An investment in real time reporting methods can greatly ease the cost of a heavy data collection burden. Even inexpensive counters are being offered with some kind of connectability. Installing these counters and programming around break points can make data collection nearly "invisible", in that the reporting can be made a part of the operation, instead of an added activity and cost.

Every manager or area supervisor would like to have a system that eliminates production reporting, one that allows the operator to work on *producing product* while the system keeps track of the inputs and outputs. With the proper training and attitude, this could probably be accomplished with stand-alone terminals. The decision to go real time, in a way, is an all-or-nothing choice. Using real time means you must program for every situation that comes up. The system is custom fit to your operation; we have found it difficult to sell this system to other 3M plants, even those with similar operations. Tieing the system in with your machinery makes the department dependent upon the system, and computer failures become very disruptive.

Not using real time or data capture leaves you short on the investment that you have made in the system. Reporting integrity and accuracy still go mostly unchecked, and the efficiency and discipline that real time can offer is lost.

Equipment choices.

Our choice of the HP1000 for this project was mostly due to the fact that our existing in-plant inventory systems were on an HP3000. We had some people in the plant who were becoming quite familiar with our HP3000 although, at the time, there was no formal systems function at our facility.

When you are contemplating a task that will have to be done almost entirely in-house, the computer of choice is... the one that you are the most familiar with! This always seems to be the biggest consideration. However, it was apparent that the HP3000 was not going to accomplish the things that we wanted this system to do.

Our choice of the HP1000 to front-end our HP3000 has worked out well, for reasons that we were aware of at the time, and several that we have discovered since then:

DS connection.

A front-end computer was always part of our plans, but much of the activity taking place on the floor would need access to the inventory information on the HP3000. File transfers between the two machines would be almost continuous. Installing DS/1000 on the front end machine solved these problems. Master/slave programs were installed on the two machines to handle the requests for information and enter transactions - with no delays.

Digital I/O card.

If the system will be monitoring the reporting integrity on a real-time basis, it needs some way to intervene in the process. The converting machinery in our tape slitting department is older, and uses relay logic to control the speeds and stops.

Upgrading this equipment to use some type of newer process controller would give us connect-ability, but that would be very costly. If we were installing this equipment to improve our process, it would make sense. The truth was that a properly trained operator can operate the process just fine.

The only things we needed to accomplish were to reset the counters and stop the machine from running during the reporting of input waste and when the computer detected a usage imbalance. The digital I/O card, along with a $20 relay for each slitter, accomplished this. Other contacts on the card are wired to the yardage counters to zero the counts. These simple functions performed by the card are all that is needed to give the HP1000 the control of the operation that it needs.

Performance and efficiency.

The HP1000 is built to interact quickly with the terminals and devices. Since our operators would be dealing with an interactive system, we wanted the response times to be as snappy as possible. This would not only enhance our productivity on the floor, but, more importantly, it would make the presentation of the system to the operator more pleasing.

With enough memory, the HP1000 has no trouble performing well, but we were able to do some things in our programming that seemed to enhance the response times. The slitters, in general, operate independently of each other, but there is shared information. Using class I/O with a couple of "server" programs allowed us to take all of the file handling routines out of the program that drives the slitter terminals. The "servers" open all of the files - or establish the DS connection - and handle requests from the slitters on demand. With the servers in place, and sharing the code segment from the terminal program, the cost (in memory) for adding another machine to the system is only a few pages for a data segment. If you design the software efficiently, most of the programs on the front end can remain in memory, and the response times are quick.

Reliability and maintenance.

The HP1000 has relatively few boards for being a mainframe computer. Our front-end A600 (with uninterruptable power and a dust-free environment) has had little downtime. Since the system was installed, we have acquired another A600 as a backup - or just for boards - and dropped our maintenance contract to labor and materials only. In 1989, with round the clock operation, our A600 front end computer had only 8 hours of down time. The cost of ownership for our HP1000s has been very reasonable - almost untouchable.

"Open-ness" of RTE.

Once you familiarize yourself with the RTE operating system, you get an appreciation for how accommodating it is to the programmer. Since our system was almost all custom, this was important.

The device drivers on the HP1000 are extremely flexible. It may not be a fair comparison, but we have had much more difficulty connecting non-HP devices to our HP3000. The availability of our HP software engineer was an asset, not only at the beginning when he was handling all of the problems we were having as novices on the system, but later on as well.

Using the open structure of RTE can be best explained by example. In the slitting operation, there are many times when one process or machine needs to pass information to another. However, the receiving program may be suspended on I/O from the terminal. A method was needed to send the message to the suspended program so it can react immediately.

A routine was built to cut into the class I/O request that is generated by the XLUEX call for the terminal driver. The routine places a buffer in class I/O, threads it onto the request tied to the XLUEX call on the terminal, then aborts the original class request. The result is that the program receives a buffer that appears as though it were typed in at the terminal. This routine is extremely handy and is easy to implement both in the sending and receiving routines.

Another commonly used strategy in RTE is storing forms in shareable EMA. This worked well in our case, because with touch screens, you will often send parts of terminal screens - touch field definitions, for example. Using EMA with the VMAIO memory-to-device data transfer was a very efficient way to manage screen painting and control.

RTE is also helpful on off-hours support. The WHATZAT, IO, and SAM programs are useful at 2 AM, when you are trying to resolve a problem over the modem.

Programming - you're on your own.

When you decide that a real time system is what you need, you may be in for some surprises. There is not a lot of software that can be purchased that can help you. If there is, it is usually in some type of a pre-packaged system that forces you to function within some type of configuration or procedure that you were trying to avoid in the first place, and your choices of equipment are limited, or you are compelled to purchase equipment that is in excess of your needs.

Our choice to proceed with the project using only in-plant programming talent(?) was motivated by a few factors. Contract programmers were available, but the cost to have one of these people live and work out in Aberdeen was high. If there was an opportunity to develop our own people by having them work on this system, we should use it. Our development team ended up being people in the plant who expressed an interest in doing this type of work, drawn from the ranks of the in-plant engineering and maintenance areas. Two of the team were trained in RTE system management and programming, and these people were joined by others who were familiar with the operation we were trying to measure in the tape area. This direction in our systems development compares favorably in cost to using outside resources - for our facility size. Although there was a lot of training expense and a learning curve for this first project, there have been several projects done since for a fraction of the cost of contracting outside help.

There is a lot that can be said for going either direction in developing a shop floor data acquisition system; to do all of the work in-house or to purchase a solution. There is a balance that must be struck between the time and expense of development and the efficiency and utility of the end product, and every situation is different.

Operator Interfaces.

Perhaps the most important decision that a project team will make will be the operator interface. The presentation of the system to the user on the manufacturing floor will have an immediate impact, and thinking ahead in this area can avoid many problems. There are many operator/user interfaces available, and the key to selecting the right one for your needs is in choosing an interface that is as accommodating to the process as possible.

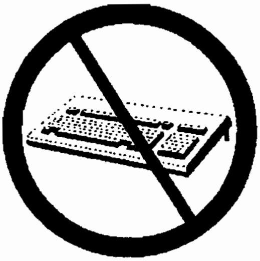
There is an operator interface that has become available in many terminal products in the last few years - the "touch" screen. This interface consists of sensing devices placed on or around the terminal screen combined with firmware to sense the position of a finger-tip touch on the screen surface. The programmer defines areas on the screen as touch sensitive and instructs the terminal to return a set response or perform a local sequence upon sensing the touch. This interface seems to work well in our application for a number of reasons:

Ease of use/training.

The touch screen can be built very efficiently, with fields labeled to match events in your operation. When an event is reached, the operator touches the corresponding field, and the program instructs them through the sequences of touches to complete the entry. The keyboard and keypad are only used if it is necessary.

Efficient.

For most of us, at least, typing is an activity that we do sitting directly in front of the keyboard, squared off. If you introduce any kind of a reach or an angle, tabbing or typing a choice becomes more difficult. A touch field, on the other hand, can be touched easily. If the operator will be frequently interacting with the terminal and moves around on their feet, this interface can be more comfortable for them.



The touch interface shines from an industrial engineering standpoint. Many companies use and are familiar with work measurement practices, or the breaking down of industrial tasks into their smallest movements. When you examine the use of a terminal and keyboard in this manner, it is a very inefficient operation. This is mostly due to the fact that keys on a keypad, for inexperienced typists, require a visual check. The other factor is eye movement between the keys and screen, a distance that is generally more than four inches.

The touch screen, on the other hand, has the finger going to the target on the screen. There is no eye movement, and the target can be made any size. Size considerations, by generally accepted practice, become unimportant after 1/2 inches of target width larger than one's finger.

You may be skeptical of the actual time savings that can be attributed to this aspect of touch. It may be more than you think, when you consider an intermittent operation on a 24 hour a day basis.

To the point.

The thing that the touch screen does best, at least in our experience with it, is keep production reporting concise. It does a very good job of eliminating the "dawdle" factor that you will find with other interfaces. When a field is touched on the screen, the program moves to the next choice, and the operator touches again. If the operator is familiar with the sequence of touches to report an item, the time actually spent at the terminal is minimal.

There are a lot of touch screen terminals on the market, and many more are becoming available. Most are built for an industrial environment (the touch screen is not nearly as useful in the office). The most important thing to look for is some type of forms storage capability. This can overcome the main drawback of touch screens - there is only so much room on the screen, and additional fields and choices usually involve repainting the screen. This is cumbersome both in terms of the time taken to switch to a new set of choices, and in supporting the programming to do so. Many of the terminals now being introduced have some sort of forms storage that allows switching to a new set of choices in the terminal memory in response to a short sequence of characters from the host computer.

From a programming standpoint, the touch screen can solve a lot of problems, especially if you are doing a custom application. In our application, almost all of the input is by touch field. The key pad on the terminal is left locked, and is opened only for numeric or bar-wanded input at specified times.

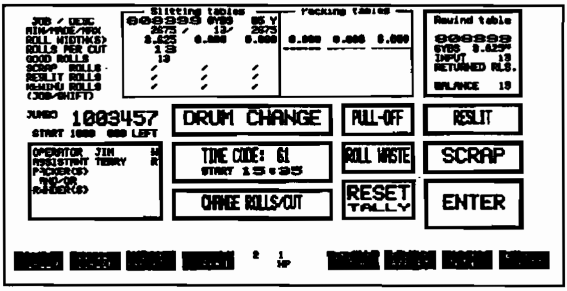
The drawback of the touch screen may be that its future is uncertain. Touch screen access is controlled differently for each vendor that supplies this interface, so access from most screen control software seems limited, although the newest release of VIEW/3000 has some touch screen control features that can be used with HP touch terminals.

Putting it all together. Some examples.

The touch terminal that we use at Aberdeen is an HP terminal Model 3082 equipped with the touch and bar wand option. It has some drawbacks in relation to its competitors; it cannot store forms, the touch fields are limited to 36 on a screen, and they do not roll with the screen display. It is a relatively new product, and the firmware lags when processing the escape codes for the terminal's special features like double size characters and touch. These can be overcome, but some programming overhead and knowledge of HP terminal operation is required.

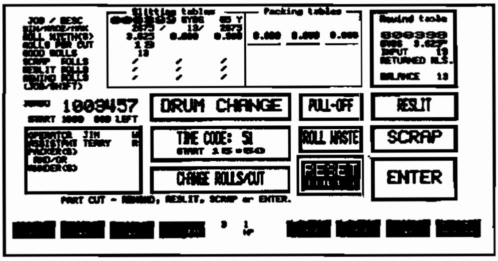
The advantages of the 3082 are that it provides escape codes to lock the configurations, key pad, and function keys on demand, and it is built like a *tank*. We have terminals with two years service that have not been touched (pun intended).

The main screen that the operator will view each day has a table for the rolls produced, organized with a column for each width (up to 3) produced on each "cut". In each column, the rolls per cut, good rolls, and the rolls for each reject roll disposition; scrap, rewind, and reslit. There are totals kept for the cumulative count for the schedule, and for the current shift.



Also displayed is the current input roll being cut, and the count for rolls being reworked and returned (This should give an indication of the amount of detail that our reporting requires). The tables are surrounded by touch fields that represent the most common events, and the 8 soft keys are used to enter transitional routines like end of schedule/shift, and information screens for job comments or to view the usage registers. This is, admittedly, a busy screen, but the use of odd size characters and line drawing gives it a more organized look. This is what the operator sees at least 95% of the time.

The system does not intervene in the slitting process except when an event is reported. The key is to tie in the reporting system with the process somehow. We accomplished this by choosing a length encoder and counter made by Durant that is inexpensive, yet has features that gave us the hook that we needed. The counter stops the machine when the required length is reached, and will not close the contact and allow the machine to run until the count is zeroed. The counter can be wired to the HP1000 to be reset to zero, and once the count is reset by the HP1000, the reset button on the counter face is disabled. As simple as this feature is, it is the key to our real time operation.



PART CUT - REWIND, RESLIT, SCRAP or ENTER.

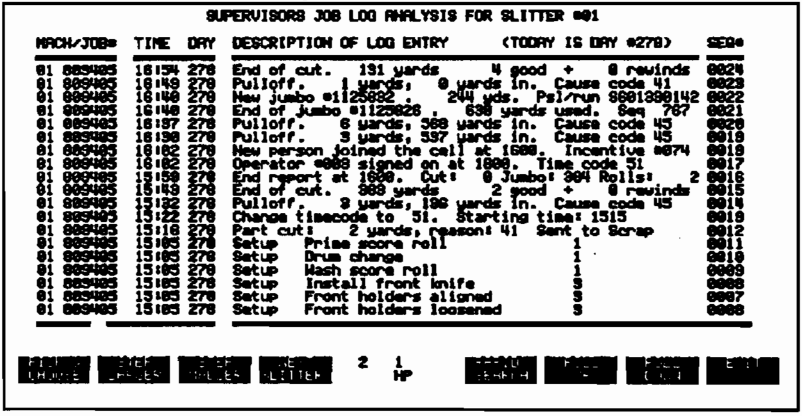
When the machine stops for the full cut, the operator will touch "reset tally" on the screen. The touch field is set to enhance inversely and return a number corresponding to that field.

First of all, the HP1000 will obtain the count from the roll tally and the count from the input tally and do the reconciliation of the input and output yards. If it does not reconcile, the operator must correct the error before continuing. Next is a check for the full yardage. If the cut is being stopped short,the program will ask for a disposition (scrap or rework), and a reason, and the yardage must go into different registers for the usage calculation. If the cut is full yardage, the amount of rolls for the cut are added to the totals, the counter is set to zero, the field enhancement is turned off, and the process continues.

Other fields on the screen operate the same way. Touching "Drum Change" invokes a routine to input a new input roll, gathering the bar wanded input from the warehouse ticket, sending the usage from the used up drum to the inventory system, then zeroing out the tally. The field containing names is touched to add or remove people from the report. Time code changes, input web waste, and roll waste all invoke their own routines and edits.

The system does not demand a lot from the operator. The reporting sequences are kept brief by using a combination of data capture with anticipation and reaction to the operators' responses. The operator does not have to do any *counting.* They only have to maintain the correct rolls per cut; the arithmetic is done by the system, and accuracy is checked with the reconciliations. The only counting is the count at the end of the report, and that is easy because it is usually a case count. This is impressive when you consider that counting and arithmetic was an activity that took fully 10% of our time. Where this is most apparent is at the end of the shift, when it was common for a machine to stop 15 minutes early to prepare their paper work. Now it is common for a machine to work up to a minute of shift change. The new operator has all of the information they need to proceed, and they can start immediately.

Each time an event is reported, an entry is made in a report log, time stamped and referenced by the job and machine number. Data stored in this detailed format is read by programs that compile the data and build entries for our corporate accounting, incentive, and waste information systems. Each program reduces the detailed data into the form and format of its client system. An example of a job log is shown here.



```
                     SUPERVISORS JOB LOG ANALYSIS FOR SLITTER #01

 MACH/JOB#   TIME  DAY   DESCRIPTION OF LOG ENTRY      (TODAY IS DAY #270)    SEQ#

 01 009405   16:54 270   End of cut.   131 yards      4 good  +   0 rewinds   0024
 01 009405   16:49 270   Pulloff.    1 yards,   0 yards in.  Cause code 41    0023
 01 009405   16:40 270   New jumbo #1125832 .   244 yds.  Psl/run 9001390142  0022
 01 009405   16:40 270   End of jumbo #1125826 .  630 yards used.  Seq  767   0021
 01 009405   16:37 270   Pulloff.    6 yards, 360 yards in.  Cause code 45    0020
 01 009405   16:30 270   Pulloff.    3 yards, 397 yards in.  Cause code 45    0019
 01 009405   16:02 270   New person joined the cell at 1600.  Incentive #074  0018
 01 009405   16:02 270   Operator #000 signed on at 1600.  Time code 51       0017
 01 009405   15:59 270   End report at 1600.  Cut:  0 Jumbo: 304 Rolls:    2  0016
 01 009405   15:49 270   End of cut.   300 yards      2 good  +   0 rewinds   0015
 01 009405   15:32 270   Pulloff.    3 yards, 196 yards in.  Cause code 45    0014
 01 009405   15:22 270   Change timecode to  51.  Starting time: 1515         0013
 01 009405   15:18 270   Part cut:    2 yards, reason: 41  Sent to Scrap      0012
 01 009405   15:05 270   Setup   Prime score roll              1              0011
 01 009405   15:05 270   Setup   Drum change                   1              0010
 01 009405   15:05 270   Setup   Wash score roll               1              0009
 01 009405   15:05 270   Setup   Install front knife           5              0008
 01 009405   15:05 270   Setup   Front holders aligned         5              0007
 01 009405   15:05 270   Setup   Front holders loosened        5              0006
```

Although the job log was designed to be a flexible solution for data interchange, it has become a favorite source of information for supervision, factory accounting, and process engineering. There are so many things that can happen during a product run, and this amount of detail is often handy in finding answers after the fact.

Data collection and "Just-in-time" Manufacturing.

At about the same time as the automated reporting was introduced into our slitting operation, there was a program implemented for "Optimized Operations", more commonly known as "just-in-time" manufacturing.

Optimized Operations is a re-thinking of production practices to move raw materials to a packed out product by performing only the work that adds value to the end product. The activities that are targeted immediately include material handling, in-process inventories, cost of quality, and data collection.

Our reporting system changed along with the department during the optimized operations project. The changes were somewhat painless because of the plant ownership of the system and the event driven nature of the reporting. Most of the reporting methods remained the same, only the activities that were altered by the new methods needed to be reworked.

Many of the changes that were implemented for the reporting system made the changes for the optimized operations project easier to carry out. Some had to do with the removal of the packaging items from the production reports mentioned before. This allowed the project team to go to home slots and a revised supply scheme without the restraints imposed by the need to count these items.

The "picture" that the system maintains for each cell and the interaction with the process that is built in for accuracy is also helpful when just-in-time floor supply is implemented. New tools like radio frequency linked terminals are becoming available and affordable, closing the distance between the floor cell and the warehouse. The material ordering and restocking can be included in the existing routines for reporting material inputs.

Just-in-time means shorter runs, quicker changeovers, smaller in-process inventories, and zero-defect quality. Having a real time system in place can have an important role in achieving all of these goals.

Summary and Conclusions.

The decision to build this system with real time and efficient user interfaces had a lot to do with the people that were involved in the project. All of us were working with the tape department and had dealt with the reporting problems first-hand. The result is a system that is a custom fit to our business.

What is remarkable is that since our project was completed, new products have been introduced that would make it possible to do it for about 1/2 the cost, and with better equipment. Going custom is becoming easier; most of the shop floor equipment has features that aid program development. There are touch screen terminals being introduced that have screen building and storage built in, which would have been helpful to us.

Having our real time system in place has allowed us to take advantage of several opportunities. Roll lengths had always been a problem, because our quality standards call for rejection and rework for any rolls short of the stated length. To compensate and be sure they were not rejected, the operators would add length to the rolls. When the terminals were installed, we were able to reduce the overlengths with a combination of better encoders and counters, and the discipline enforced by the usage calculation done by the program. The savings from this overlength reduction have been enough to pay for the project over again.

What the most important consideration to ensure that a shop floor system will succeed? I believe it is the presentation of the system to the operator who will be using it. People who are involved in production generally measure their performance on the job by *what they produce*, even if they are not covered by piecework incentive, as the Aberdeen operators are. Cumbersome interfaces, detailed manuals, or poor response times can sidetrack a project quickly. Some people respond to change and training better than others; the key is to disarm these problems by making the reporting independent of who is doing the work.

Our experiment and investment in real time, for the most part, is viewed as a success. It is the first such attempt in 3M company, and it was watched very closely. It has become ingrained in our operation and adds value to our process. The success of the project may have been due to the nature of the business that it was built for as much as the project itself. Time will tell, as the number of projects of this type and the vendor offerings grow.

TITLE: A Read and Write Interface From HP-UX/4GL to
HP Image

AUTHOR: Aki Kanebo
Montgomery Laboratories
555 E. Walnut Street
Pasadena, CA 91109

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 1015

Transaction Processing Monitors and Database Systems

Paul McGuckin, Oracle Corporation
20 Davis Dr.
Belmont, CA 94002
(415) 598-8000

Transaction processing monitors (TPM's) have been used for years in mainframe computer environments to increase the efficiency of database processing. IBM's CICS is best known example of a mainframe TPM. However, until very recently the UNIX operating system has not enjoyed the benefits of TPM's. During the last three years two developments have occurred which are paving the way for TPM's to become widely available in the UNIX arena:

- X/Open's standardization efforts in the area of Distributed Transaction Processing
- AT&T's development of a TPM based upon the X/Open standard

X/Open and the XA Standard

X/Open is a worldwide organization consisting of representatives of computer hardware manufacturers, software developers and end users. The goal of the organization has been to meld *de jure* and *de facto* industry standards into an interface which will provide for the portability of software across hardware platforms. Their most original work has been in the area of Distributed Transaction Processing, which is called the XA model. This model specifies the relationship between the Application Program (AP -- the requestor of services), the Resource Manager (RM -- the provider of services) and the Transaction Manager (TM -- the co-ordinator of services, which I have above called the TPM).

Figure 1 illustrates the model envisioned by X/Open. The AP communicates with one or more RM's to request services. The AP informs the TM of the beginning and end of any transactions which will be initiated with the RM. This notification is necessary because it is the job of the TM to co-ordinate transactions, especially when the transaction may encompass work from more than one RM. The TM and RM's communicate regarding the initiation and completion of transactions.

(A transaction is an atomic unit of work, which is either realized or aborted as a single unit. Thus, in a database system multiple updates to a table are not permanent until a COMMIT is issued; when the COMMIT is issued either all the rows affected will be updated or none of them will be (if there is a system failure or serious error.) If a single transaction includes updates to tables on multiple systems, a two-phase COMMIT is required to guarantee that the transaction is atomic. In a two-phase COMMIT all participating sites must PREPARE TO COMMIT; if this phase is successful then the final COMMIT is issued by the commit co-ordinator. In the XA model the TM is the commit co-ordinator.)

X/Open has released a preliminary specification for the XA interface, which contains the functions to be used for communication between the TM and the RM's.

AT&T System/T

Over the last few years, AT&T has developed a TPM named System/T, the first (and so far the only) TPM available under the UNIX operating system. Version 3 of System/T requires AP's and RM's to reside on a single computer system (although at least one third party has developed a version which supports locating the AP's on remote machines). Version 4 of System/T provides full communication support for distributing AP's and RM's across a TCP/IP network. Version 4 of System/T is also compliant with the preliminary X/Open XA specification.

Oracle's Integration with System/T

During 1989 Oracle's UNIX Product Division developed a version of the Oracle RDBMS which uses the System/T TPM. This development work was accomplished jointly with Pyramid Technology and Independence Technologies, an Oracle VAR.

Figure 2 illustrates the architecture of standard Oracle under UNIX. Each database client (an Oracle tool or embedded SQL program) is connected to its own Oracle server process. Server processes have access to Oracle's Shared Global Area, which contains database buffers, redo log buffers and co-ordinating data structures. Historically, the one-server-per-client model was implemented to provide for efficient utilization of symmetric multi-processor (SMP) computer systems, by allowing the operating system to schedule servers across all available processors. This architecture has resulted in very high transaction rates for SMP systems.

Multi-Stated Server

In order to improve transaction throughput, and in particular to support many more clients than formerly practical, a multi-stated version of the Oracle server was designed and implemented. This Multi-Stated Server (MSS) is capable of serving the requests of more than one client. Because we can now reduce the number of servers in a system, both the memory and process overhead associated with an instance of Oracle are thereby reduced, resulting in more efficient processing.

In order to implement MSS, a number of changes were made to Oracle's architecture under UNIX:

Data structures: Oracle stores most session-specific information in a data structure known as the Program Global Area (PGA). In the conventional one-server-per-client architecture this data is private to each server process. In the MSS model, the PGA now resides in the Shared Global Area (SGA), where it can be accessed by any available server. Likewise, the multiple execution stacks of the MSS (one for each client being served) are mapped into the SGA. Finally, all context areas (cursors) which have been allocated also reside in the SGA.

SQL*Net: Oracle's SQL*Net protocol was modified to include a unique client identifier with each message sent, so that the MSS knows which client sent the message.

By virtue of these architectural changes, the MSS is able to perform processing on behalf of multiple clients. When a client transmits a SQL*Net message to the MSS, the MSS first determines which client is making the request by examining the client identifier contained in the SQL*Net message. The MSS then maps in the PGA, execution stack and any cursors allocated on behalf of that client. Processing then continues in the conventional manner.

Note that clients are not tied to specific servers: since all servers can access the SGA, the each client request can be routed to the MSS which is least busy at that particular moment. The routing of requests is handled by System/T, which we will examine next.

Integration with System/T

Figure 3 illustrates the roles of System/T and MSS. When an Oracle client session is started, it is automatically connected to System/T. This connection is transparent, requiring no special action on the part of Oracle clients. System/T then routes each SQL*Net message from the client to the MSS which it determines is least loaded. In this manner, System/T is responsible for load balancing work across all available servers. On the return trip, System/T examines client identifier in the SQL*Net message received from MSS and routes it to the appropriate client.

In addition to message routing and load balancing, System/T is responsible for providing the following services:

Server management: System/T dynamically creates and terminates MSS processes, to satisfy the current demand for database services. In addition, System/T provides additional fault-tolerance by automatically restarting any server which terminates abnormally.

Transaction monitoring: System/T provides facilities which monitor transaction activity and system utilization. This information can then be used by the Database Administrator to adjust priorities, queues and resources to maintain the required service level.

Future Directions

Continuing development work on Multi-Stated Server and System/T integration is focused in the following areas:

Two-Phase Commit Co-ordination: Oracle Version 7 supports two-phase commits, which allows updates to two separate databases to be committed as a single atomic transaction. System/T can then serve as the commit co-ordinator across multiple Oracle databases. This transaction model conforms to the X/Open XA Distributed Transaction Processing Standard, further evidence of Oracle's commitment to standards.

Multi-client Transactions: Currently, only a single client can contribute to a transaction. This enhancement will allow one client to initiate a transaction, one or more clients to add to the transaction, and yet another client to commit the transaction. For example, this capability could be utilized in a telesales system, where the initial client adds a new customer record and passes the customer on to a specialized sales consultant. The consultant assists the customer in placing the order. The customer is then passed to the credit department, which enters additional information into the database and commits the transaction.

Foreign Server Support: System/T is not limited to managing Oracle servers, but may concurrently manage foreign servers. For example, a transaction may include updating the database, running a report (which might be checks or invoices) and printing that report. A print server may be responsible for printing that report. System/T can manage not only the database aspects of the transaction, but also insure that the transaction is not considered complete until the report is actually run.

Conclusion

The integration of database systems with TPM's under UNIX provides significant benefits to large transaction processing systems. By reducing memory usage and process overhead TPM-based database systems can support many hundreds of database clients while ensuring consistent response times.

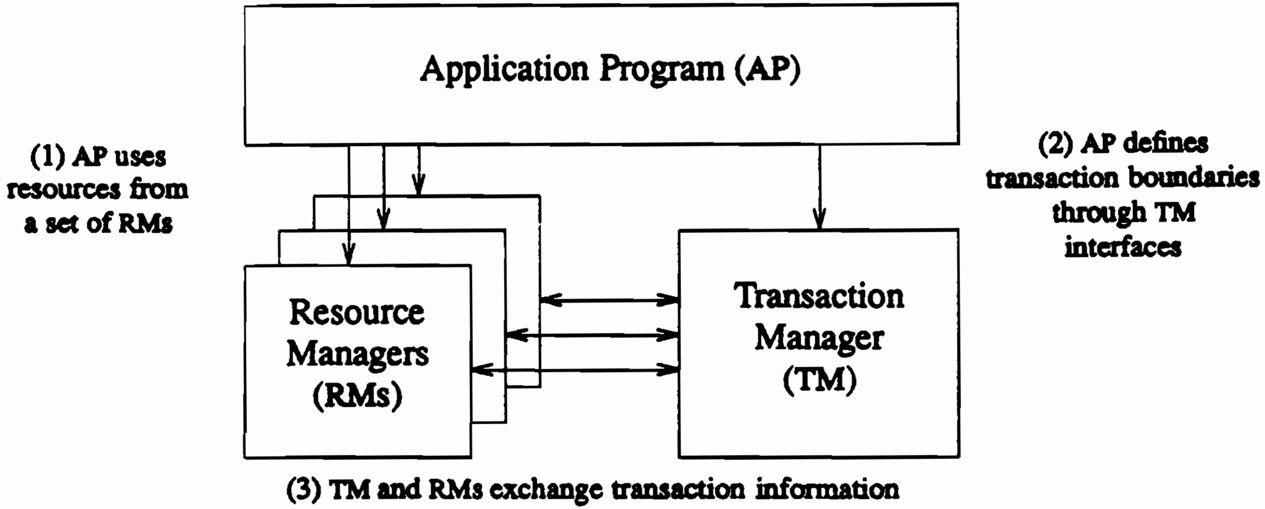
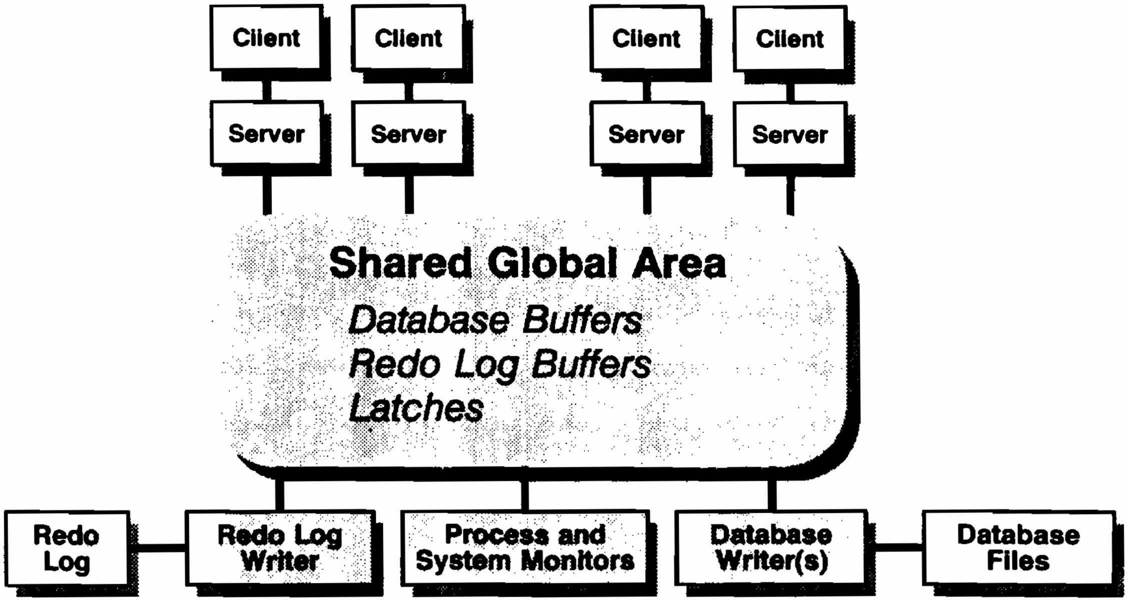

Application Program (AP)
(1) AP uses resources from a set of RMs
(2) AP defines transaction boundaries through TM interfaces
Resource Managers (RMs)
Transaction Manager (TM)
(3) TM and RMs exchange transaction information

FIGURE 1

ORACLE Version 6 for UNIX



Client

Client

Client

Client

Server

Server

Server

Server

Shared Global Area

Database Buffers

Redo Log Buffers

Latches

Redo Log

Redo Log Writer

Process and System Monitors

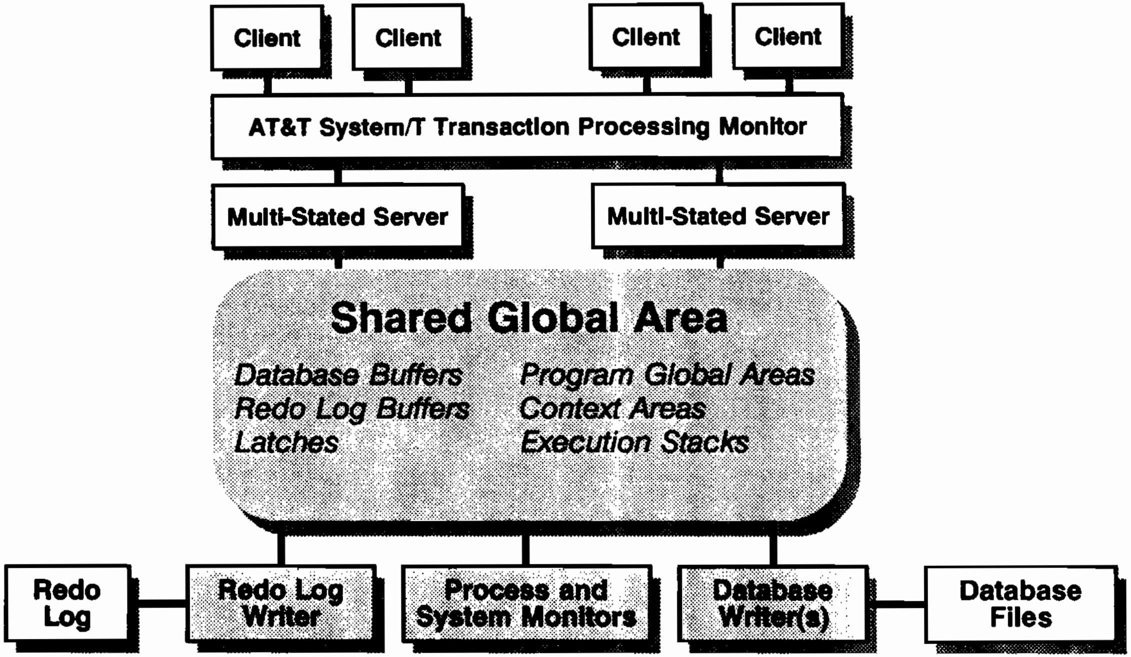
Database Writer(s)

Database Files

FIGURE 2

ORA167.03

Integration with Transaction Processing Monitors



ORA167.08

FIGURE 3

TITLE: Developing Commercial Applications For UNIX

AUTHOR: Colin Bodell

Micro Focus

2465 E. Bayshore Rd., Suite 400

Palo Alto, CA 94303

415-856-4161

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 1017

Transferring CPLOT Graphics from the HP 1000 to a PC

by Gerald Lisowski and Dan Kukla
ICI Americas, Inc.
Western Research Center
1200 S. 47th Street
Richmond, California 94804
415-231-1390

In the PC world HPGL has become a *de facto* graphics standard. Most graphics plotters accept HPGL input and most graphics programs can produce HPGL output. Recently high end word processors, such as WordPerfect by WordPerfect Corp. and Word b, Microsoft, have acquired the ability to import HPGL graphics. They can import the graphics either directly (Microsoft Word) or through a conversion program (WordPerfect).

Incorporating graphics into our documents is an important feature. Before our word processors obtained this ability, the only work around was to create the graphic separately and leave space for it in the document. If the graphic occupied space in the body of the text we would usually paste the graphic and document together and photocopy the resulting page. This increased the time and effort needed to produce a document. If something was later added or removed, it usually meant the whole process would need to be repeated, typically with a lot of aggravation. If the graphic occupied a whole page, things were a little bit easier. But even with full page graphics we often need to add captions, running heads, or page numbers. We still had to go through the paste and photocopy routine. Now we let the word processor handle the layout details. It does it faster and smoother than we ever could.

One type of graphic we are especially interested in is chromatograms. When a chemist injects a mixture of compounds into a machine called a chromatograph, the chromatograph separates the compounds and sends them to a detector. The detector sends the output to a device capable of accepting its signal. If the device is a recorder a chromatogram is obtained. A chromatogram plots detector response on the y-axis verses time on the x-axis. The chromatogram contains two valuable pieces of information, the retention time and peak area. The retention time is how long the compound spent in the chromatograph, and can help the chemist identify the compound. The peak area is proportional to how much of the compound was placed in the chromatograph. Using this information the chemist can quantitate how much of the material was in the original sample. Figure 1, supplied by HP as part of its Laboratory Automation System (LAS) training course, is an example of a chromatogram. Figure 2 is the same chromatogram with the baselines drawn in after processing by LAS.

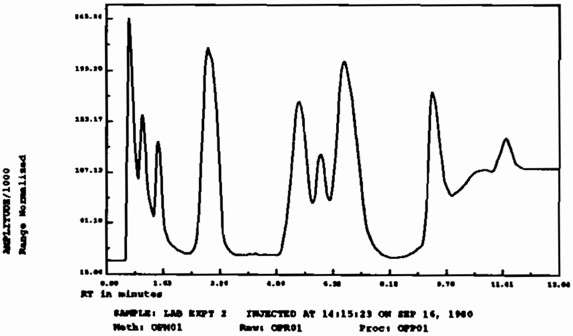


Figure 1. Raw chromatogram

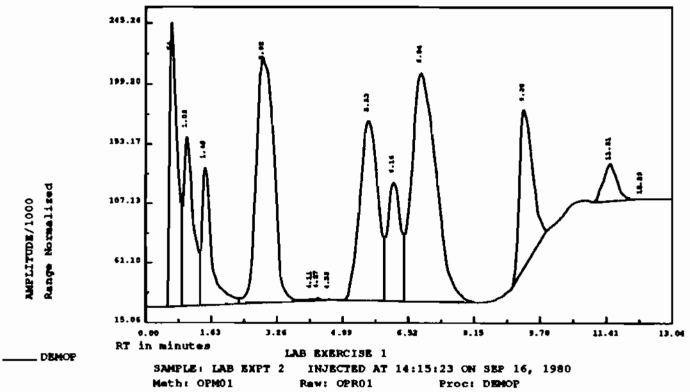


Figure 2. Processed chromatogram

A recorder is not the only device capable of accepting the signal from a chromatographic detector. The signal also could go to a computer with the hardware necessary to receive the signal, and the software necessary to process it. Such a system is called a chromatography data system. Given that HP is a leader in analytical instrumentation and computers, it's not surprising that they make a chromatography data system. In fact they make several. One family of these, the LAS systems, are based on the HP 1000. The system used on an E/F series computer running RTE-6 is the 3357 while the system used on an A series computer running RTE-A is the 3350A. Both units can store the raw data points to allow for subsequent reanalysis and replotting of the data.

One program that can replot the data stored in a 3357 or 3350A is CPLOT from Hewlett-Packard[1]. CPLOT uses Graphics/1000-II and can take the chromatographic data and replot it on a graphics terminal, or send the plot to various graphics printers or plotters. Two of the supported devices are the HP7475 and HP7550, both HPGL plotters. The output also can be sent to a spool file. Using CPLOT we can send the output to a spool file that CPLOT thinks is an HP7475 or HP7550. The spool file is a text file that can be sent to a PC disk using a variety of programs. The program we use is Reflection 7 by Walker Richer & Quinn, Inc.

Preparing CPLOT

On both RTE-6 and RTE-A systems, file "USERC needs to be set up correctly before CPLOT can be used to produce an HPGL file. It must contain spool table entries for at least one of the two supported HPGL plotters. The text below was taken from a "USERC file set up to allow the spooler to emulate either the HP7475 or the HP7550.

```
!SPOOL
 spool table - line 1=no. of entries, each entry on separate line:
 wsp name, plot limits x1,x2,y1,y2,direction code,aspect ratio
==>2
W7475,0.,257.8,30.,198.1,250,.6232
W7550,0.,253.0,30.,195.0,250,.6180
```

Information on what the entries mean and how to modify "USERC can be found in the CPLOT manual.

You also may need to modify the pen section of "USERC to have all the pens be 1. When Microsoft Word prints an incorporated HPGL file, the characters drawn with pens other than 1 are fuzzy. The characters drawn with pen 1 are nice and sharp.

If CPLOT is run from CI, the work station programs (wsp files) must be in the PROGRAMS or working directory.

Producing the HPGL file on the 1000

The first step in getting the PC graphic is to create the HPGL file on the 1000. Two procedures can be used, interactive and batch .

Interactive mode

The interactive mode is available only with the RTE-6 version of CPLOT. It's more involved than the batch mode, but also more flexible.

The first thing is to set up the spool file that will contain the HPGL commands. Log on with a capability of at least 30. Next use the CR command to create a type 3 or 4 file to hold the HPGL commands. With the CR command you must specify a size for the file. A size of 24 is suggested. This is the default size for scratch files, and the *Programmer's Reference Manual* recommends it for user created files. If more file space is needed the system will automatically create it. Next issue the command RP,SMP, if SMP has not already been RP'd. Finally issue the command SL,lu,filenamr,WR. LU must be in your SST and should not be in use. Filenamr is the name of the type 3 or 4 file created above with the CR command. You now are set up to redirect output from an lu into a file.

Next run CPLOT. In CPLOT select your raw file and any other parameters you wish to modify. If necessary, you can modify, plot, and analyze until satisfied with the results. When all the parameters are set press F4, the menu function key. Then press F5, the plotter function key. In the plotter menu, press F6, and type the number of the spool LU. Then press F7 to choose the plotter. Type W7475 or W7550, whichever is in the spool table in "USERC. Plot to the spool file by pressing the plot function key, F5. After plotting is complete press F4, the menu function key. Finally press the end key, F8, to exit, and answer yes to the "*ARE YOU SURE YOU WANT TO QUIT*" question. At this point you will be in graphics mode. If you are using Reflection, get back to alpha mode by pressing alt-7 on the numeric key pad.

Once you are back into RTE type CS,lu,EN to close the spool file. If you do not want to keep SMP around, and you have the capability, type OF,SMP.

Batch mode

Batch mode can be used with the RTE-6 version of CPLOT, and is the only way to get an HPGL file from the RTE-A version. As in the interactive mode, the RTE-6 version of CPLOT requires setting up a spool file to contain the HPGL commands. This is done using the same commands described above for the interactive mode.

Spooling HPGL output on RTE-A systems requires that the appropriate plotting device is present in the system, and its LU is below 64. A dummy LU can be generated for plotting purposes. The following is an example of an Interface and Device Table entry for an RTE-A answer file to generate a dummy LU (LU 9) to support plotting to HP7475 and HP7550 plotters:

```
    IFT,%ID*37::RTE_A,SC:27B
    DVT,,,TO:2000,DT:77B,TX:0,DX:1:36B,PR:0,LU:009
```

Turn spooling on to redirect the output from the LU to a file by entering SP,ON,LU,FILENAME, where LU is the number of the plotter LU and FILENAME is the name of the HPGL file you want to create.

After the LU is set up to direct its output to a spool file, generate the plot by running CPLOT in batch mode. On an RTE-A system the spooling (SP) option must be used along with the codes for adding retention times, names, and baselines to the plot. This is done by issuing the command CPLOT,WSP,LU,,RESULTFILE,,SP... at the system prompt. The run string parameters are described in the CPLOT manual.

As an example, on an RTE-A system the commands

```
SP,ON,9,PLOT.PLT
CPLOT,W7475,9,,DATA.RES,,SPBL
```

will redirect output from LU 9 to the file PLOT.PLT and generate a plot emulating an HP7475 based on the chromatographic data in the file DATA.RES. The plot will have baselines drawn in.

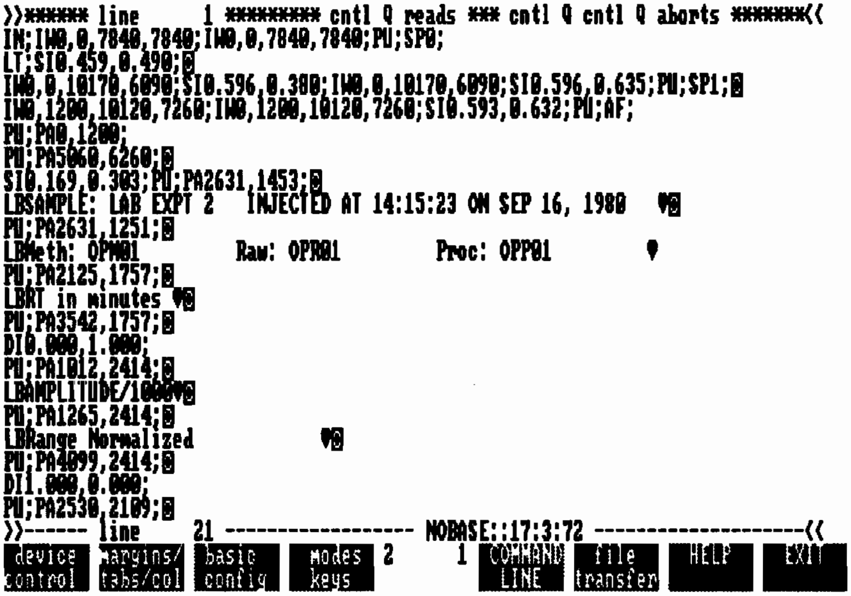
On an RTE-6 system the spooling option must not be used. For an RTE-6 system the equivalent set of commands is

```
CR,PLOT::-17:3:24
RP,SMP  (if SMP is not already RP'd)
SL,33,PLOT::-17,WR
CPLOT,W7475,33,,DATA,,BL
```

On an RTE-6 system the spool file should be closed after the plotting is finished. As in the interactive mode type CS,lu,EN. On an RTE-A system the spool file is automatically closed when the plotting is completed.

Transferring the HPGL file to a PC

You now have the HPGL file on the 1000. How do you get it to the PC? If the HPGL file was a normal text file the answer would be simple. Do a LOG BOTTOM with the TO DEVICE set to TO DISK. Then copy, store, or dump the file to LU 1. Unfortunately this won't work with HPGL files. HPGL files are not normal text files. To see why, run EDIT on the HPGL file. If you list it out it looks like a regular text file. However if you go into screen edit mode you get something like figure 3.



```
>>****** line      1 ********* cntl Q reads *** cntl Q cntl Q aborts *******<<
IN;IW0,0,7840,7840;IW0,0,7840,7840;PU;SP0;
LT;SI0.459,0.490;▣
IW0,0,10170,6090;SI0.596,0.380;IW0,0,10170,6090;SI0.596,0.635;PU;SP1;▣
IW0,1200,10120,7260;IW0,1200,10120,7260;SI0.593,0.632;PU;AF;
PU;PA0,1200;
PU;PA5060,6260;▣
SI0.169,0.303;PU;PA2631,1453;▣
LBSAMPLE: LAB EXPT 2   INJECTED AT 14:15:23 ON SEP 16, 1980   ♥▣
PU;PA2631,1251;▣
LBMeth: OPM01           Raw: OPR01          Proc: OPP01          ♥
PU;PA2125,1757;▣
LBRT in minutes ♥▣
PU;PA3542,1757;▣
DI0.000,1.000;
PU;PA1012,2414;▣
LBAMPLITUDE/1000♥▣
PU;PA1265,2414;▣
LBRange Normalized            ♥▣
PU;PA4099,2414;▣
DI1.000,0.000;
PU;PA2530,2109;▣
>>------ line    21 ------------------- NOBASE::17:3:72 --------------------<<
```

Figure 3. EDIT screen of an HPGL file.

The funny looking characters[2] at the end of some of the lines are control characters that are an integral part of the HPGL file. In Reflection a simple LOG BOTTOM strips out these necessary characters. A way must be found to transmit these characters to the PC. In Reflection 7 this is done with the CAPTURE command.

The CAPTURE command passes through all characters. The following command file can be used to transmit an HPGL file to a PC.

```
DISPLAY "What is the host filename?"
ACCEPT V1
DISPLAY "^M^J"
DISPLAY "What is the PC filename?"
ACCEPT V2
DISPLAY "^M^J"
TRANSMIT "**"
OPEN $2
SET CAPTURE YES
TRANSMIT "^M"
WAIT FOR "^Q"
CLOSE DISK
SET CAPTURE NO
```

Where ** is DU,$1,1 for an RTE-6 system and CO,$1,1 for an RTE-A system. The HPGL file is now ready to be used by MS-DOS programs.

Using the above command file causes some header and ending "junk" to be introduced into the PC file. This seems to present no problems, either to programs that use the file directly, such as Microsoft Word, or programs that translate the file into their own format, such as Word Perfect.

One fact needs to be mentioned about the HPGL file produced by CPLOT. It contains a lot of space around its border. Figure 4 is a directly imported HPGL file with a box around its perimeter. It is the file used to produce figure 1.

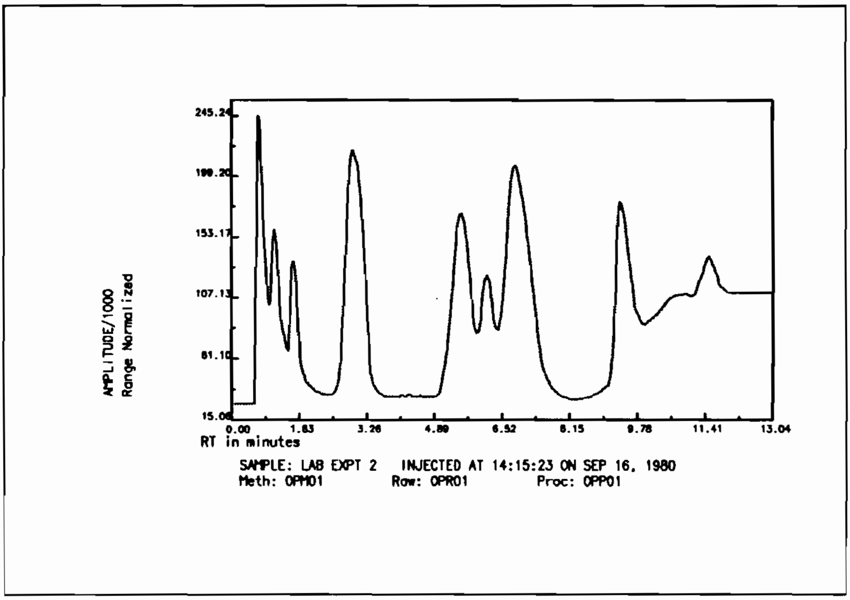


Figure 4. Directly imported HPGL chromatogram

You can see that there is a lot of blank space around it, especially at the top and bottom. If you use a program like Word Perfect, with its own graphics editing, this is only a minor annoyance. On the other hand if you use a program like Microsoft Word, with no graphics editing capabilities, you probably will want to use another program to edit out the superfluous blank space.

Although we have described using CPLOT to generate the HPGL file and Reflection 7 to get the file to a PC, the techniques we have demonstrated can be adapted to other programs. Any HP 1000 program that can send graphics to an HPGL plotter should be able to have its output redirected to a spool file. Any good terminal emulator should allow you to copy the HPGL file from the HP 1000 to your PC. Just remember to make sure that it passes control characters as well as normal alphanumeric characters.

<u>References</u>

1. Hewlett-Packard sells and supports CPLOT for the HP-3350A. It no longer sells CPLOT for the HP-3357. If you wish to obtain CPLOT for an HP-3357 contact your sales rep.

2. Figure 3 was taken from the screen of a PC with a Hercules compatible graphics card. The control characters are shown in their IBM character set representation. HP terminals display the control characters differently.

Adventures in Airborne Real-Time Data Acquisition

Dean Clamons and Mary Peters
Naval Research Laboratory
Washington, DC 20375
(202) 767-2384

Abstract. The Naval Research Laboratory has developed an HP-9000/835 based system for airborne geophysical measurement. This system is an adaptation of an earlier system which ran under RTE on HP-1000 systems. This paper discusses our experience in using HP-UX for building real-time data-acquisition systems. The system's operator interface is a menu system designed for alphanumeric terminals. We show some techniques for quickly building menu driven systems written in C. The inter-process communications and interrupt handling facilities of HP-UX were used extensively in developing this system. Our experience in using these capabilities are detailed. The system acquires several streams of data through RS-232 Multiplexor and Asynchronous FIFO Interface cards. Techniques for using these interfaces for data acquisition, as well as some problems, are discussed. We conclude with results of recent flights with the system and recommendations for development of real-time data acquisition systems.

1. Introduction.

The Naval Research Laboratory has developed an airborne system for the measurement of the earth's magnetic and gravity fields. The system was originally developed to make measurements over open ocean, but is being extended to make measurements over land as well. The system was developed using an HP-1000 as the main data collection computer. The inputs required to calculate the local magnetic and gravity variation from the nominal field value are position, velocity, vertical and horizontal acceleration, magnetic field, gravity, and time. The system evolved from one which simply recorded data during the flight to one which records and displays data during the flight. RTE was well suited for this type of real-time data acquisition system, but the HP-1000 computers which we were using were becoming less reliable and their speed was inadequate for some of the real-time and post acquisition processing which we wished to perform. We decided to move to higher powered computers, and chose to use HP-9000/835 computers. We knew that this move would require redesign of some of the custom interfaces used by the system and an extensive rewrite of the system software, but felt that the expense would be justified by a more capable and flexible system.

2. Description of the Project.

2.1. Purpose of the system.

The goal of the NRL Airborne Geophysical Sensor System (AGSS) project is to develop the techniques of airborne surveying for rapid, broad surveys over oceanic regions and other remote areas inaccessible by ship or land vehicle. The data we have collected while developing this system has been used to study the ocean floor and the underlying geological structure. The airborne platform for this system is the NRL P-3 Orion, which has been modified for scientific research use. In addition to the military air crew and civilian scientists, approximately one dozen people per flight, the plane can carry up to 3500 pounds of equipment and has an operational range of up to 2500 km per flight. While in operation with the older HP-1000 computers, this system has been used to collect magnetics and gravity data in the North Atlantic, South Atlantic and off the peninsula of Antarctica. Since the transition to the HP-9000/835, the system has been flown over the Eastern Shore of Maryland, the Chile Rise in the Pacific, and the Indian Ocean.

2.2 Hardware description.

The AGSS hardware consists of a number of sensors connected to the HP-9000/835 computer through the 27140A Asynchronous 6-Channel Mux card and the Asynchronous FIFO Interface (AFI) card. Figure 1 shows a block diagram of the hardware configuration. Since this is an experimental system, the exact configuration varies from flight to flight. Some configurations, for instance, will have two Mux cards to accommodate more serial devices.

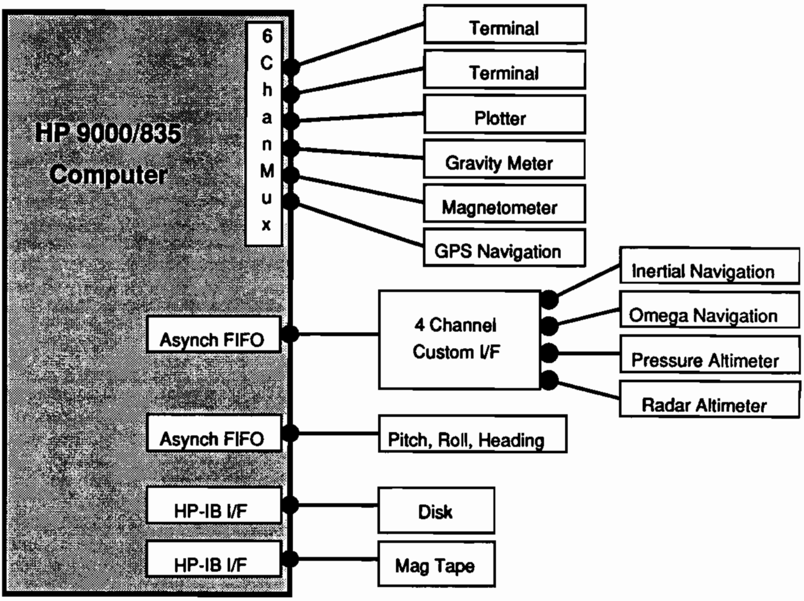


Figure 1
Hardware Block Diagram

The console terminal is an HP 2623A, and the second terminal is usually a PC. The plotter is a Calcomp 1023, which emulates an HP 7585 plotter. The disk is a 600 Mbyte HP 7963B, and the tape drive is an HP 7980. All data is stored on disk files during the flight and copied to mag tape at the end of each flight. The disk size allows all data for a particular project to be stored on the disk for ease of processing during a field trip.

The magnetometer is an EG&G Geometrics 813 unit. It samples the magnetic field at rates controlled by switches on the magnetometer. The computer can request a data value, as well as the switch settings through an RS-232 link.Typically we read the magnetometer at a rate of one data value every 2 to 3 seconds.

We have three sources of navigation because they serve different purposes. The Global Positioning System (GPS) provides very accurate position and velocity data, but until all of its satellites are in place it is usable only a few hours per day. Most flights are scheduled for periods of good GPS coverage. The Omega and inertial navigation systems operate all of the time, but they are of lower accuracy. The inertial system exhibits low fix noise but drifts over the course of a flight while the Omega does not drift but has high position fix noise. These two in combination can provide quite good position information, though not as good as the GPS.

GPS position and velocity is provided by a TI 4100 Navigator. It continuously outputs data on an

RS-232 line. The data consists of blocks of formatted binary integer and floating point data. The amount of data is between 500 and 1000 bytes per second. Since the data comes in bursts at nearly random times, the buffering capability of the Mux card is important.

The Omega receiver and the inertial navigator, both manufactured by Litton, provide data over an ARINC 561 bus. This is a synchronous serial bus which we have converted to 32 bit parallel data with a custom built interface. The inertial navigator supplies about 170 32-bit values per second, and the Omega receiver supplies about 56 values per second. Both supply data in BCD format.

Altitude is measured by two independent methods. The primary altimeter is a pulse radar unit built at NRL. It pings at a selectable rate between 10 kHz and 40 kHz. The readings are averaged by hardware to produce 128 samples per second. Since some pings are missed due to noise, a poor reflecting surface, or plane attitude, there is a counter which counts at a 10 kHz rate which is recorded with each averaged value. This gives us an accurate measurement of the time for each altitude. This is important since the altitudes are doubly differentiated to calculate vertical acceleration. The second altitude source is a pressure gauge. This sensor provides a good measurement of local changes in altitude, but it is affected by local weather conditions. It's main advantage is that it is not affected by the nature of the surface below the aircraft. It is used to fill in information when the radar altimeter is too noisy. The pressure is read at a rate of 100 Hz.

In some applications the attitude (pitch, roll, and heading) of the aircraft, as well as inertial accelerometer measurements, are recorded using some of the outputs of the inertial navigator's stable platform. These outputs are in the form of synchro signals and pulse trains which are interpreted by a special interface built at NRL. They provide about 5500 bytes of data per second.

We usually carry two computers on board so that we have a backup (many of our field trips are to remote parts of the world). In order to provide some extra processing power during flights, we network the two computers together so that the spare computer can be used for processing. One of the problems with UNIX as a data acquisition operating system is that it can lose data if shut down improperly. Since power on the aircraft is not highly reliable, we decided to provide uninterruptable power supplies for the computers and disks. These are essentially batteries which have enough capacity to keep the systems running long enough to do an orderly shutdown.

2.3. Software requirements.

Since we had already developed an RTE based system for AGSS, most of the algorithmic issues had been resolved. Because of the experimental nature if the system we continue to tweak the scientific algorithms, but the main emphasis during the conversion was to adapt the system to HP-UX and to provide a better user interface to the system. Of course, the most important aspect of the system is that it must collect all of the data during the flight with little or no data loss. In addition, we wanted the system to be flexible and extensible in order to accommodate the ever changing nature of an experimental system.

The RTE version used the question and answer method of communicating with the system, and it had a monitor program which displayed several values which indicated how the system was operating in real-time. We decided to use a menu driven operator interface in the HP-UX version of the system. The overall structure of the system was taken from the RTE version. Figure 2 shows a block diagram of the software structure. There is a driving program, *agss*, which is responsible for starting and stopping all of the other programs. The program *params* is run by *agss* when the operator wishes to alter operating parameters. There is a separate program responsible for reading each data stream. These programs are started up by *agss*, and they communicate with *agss* through shared memory and message passing. This structure allows the system to continue running if one input device or program should fail. All data is stored on disk files, one file for each data type. All of the data is time tagged so that off-line processing programs can properly combine the data.

The system is started by the operator running *agss*. *Agss* then runs *params* to get system initialization parameters. Then *agss* runs the programs which the operator requested. It sends each of them a message queue id which they can use to communicate problems to *agss*. It also tells the acquisition programs where to put information in shared memory. The acquisition programs will normally continue to go through a data acquisition and storage loop until the operator requests the system to stop. All data is stored on disk, and some of the data is stored in shared memory where *agss* can display it for the operator. Each of the acquisition programs intercepts all interrupts so that they can inform *agss* of any unexpected problems.

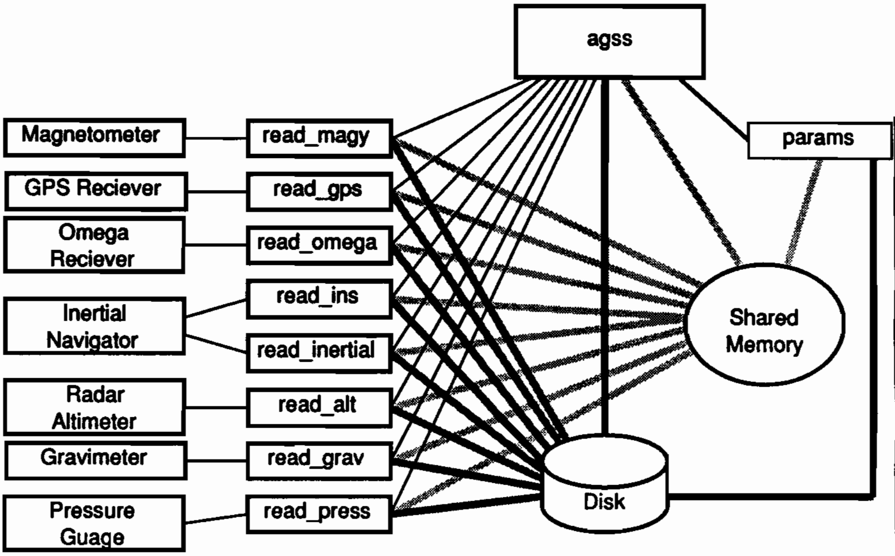


Figure 2
Software Block Diagram

In addition to the programs shown in Figure 2, there is a set of programs which test each of the input devices. The test programs check that each device is operational and, to the extent possible, check that all of the lines on the cables are connected. They do this by checking that both ones and zeroes are found on each bit of the data. All of the data files are binary in order to conserve space and time. Therefore, there is also a set of programs which list the data in the various types of data files. These listing programs can also be used to convert the data into ASCII files for processing and editing.

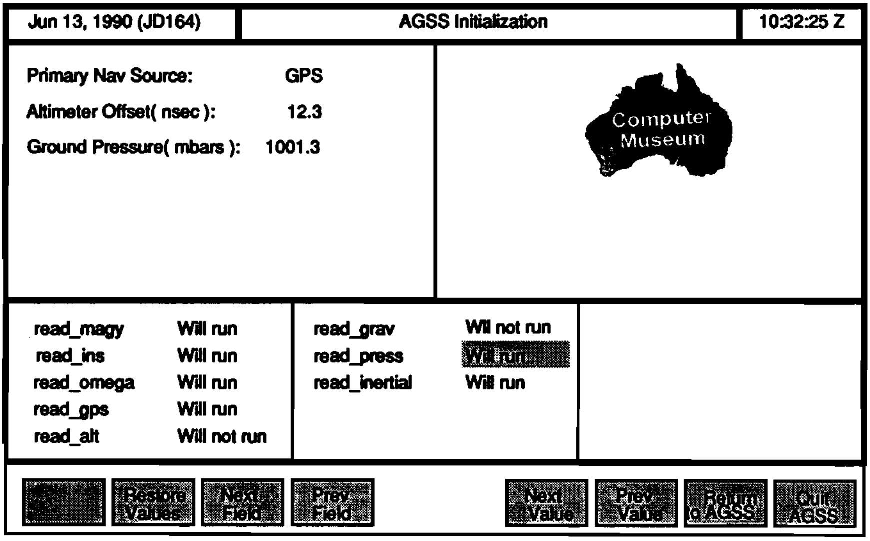
3. Operator Interface.

3.1 Operator's view of the Interface.

The nature of the operator interface was determined partly by our experience with the earlier RTE system and partly by the equipment available for the system. Earlier experience indicated that a menu driven system was highly desirable, but since we did not have a raster graphics display on the system we could not use a graphic user interface such as X Windows. We decided to use a function key driven menu system which was modeled roughly on HP's terminal configuration method. The operator interface serves two purposes. First it allows the operator to tell the system which parts of the system are to be active and to tell it the values of some operating parameters which the system cannot measure. Second, it displays the status of the system and shows the values of some of the measured quantities. One of the drawbacks to the RTE system was that the operator could not alter all of the system configuration during flight without stopping the program and re-starting it. Indeed, some configuration changes required re-loading the program. The new operator interface required that all configuration parameters be alterable in mid-flight without disturbing data acquisition.

Figure 3 shows a typical initialization screen. This screen appears at system start-up time as well as whenever the operator asks to change any operational parameters. The operator moves from field to field on the screen by using the function keys or by using the arrow or tab keys. The field being operated on is highlighted. The changeable fields on the screen are of two basic types, enumerated and text. Enumerated fields have a small number of fixed possible values, and the operator may scroll among them using the function keys. The highlighted field in Figure 3 is an enumerated field whose value may be "will run" or "will not run". Text fields are fields which require the operator to enter a value. They may be limited to numeric values or may be alphanumeric. The "Altimeter Offset" field in Figure 3 is an example of a text field which may only take on a numeric value. The values entered by the operator do not take effect until the "Return to AGSS" function key is pressed. Therefore, the "Restore Values" key can be pressed at any time to restore the values in effect when the screen was called up.

The upper portion of the initialization screen is used to enter parameters needed by the various acquisition and display programs. The lower portion of the screen is used to determine which portions of the system will be active, i.e. which data will be acquired. At present the parameters which appear in the top portion of the screen do not depend on which programs are selected to run in the bottom portion of the screen. For instance, the "Altimeter Offset" field is displayed even though the operator has chosen not to run the altimeter program (read_alt). This is perhaps not the ideal situation from the operator's point of view, since he may not be familiar with the radar altimeter. However, this is a minor problem since whatever value is entered here will be ignored by the system if the altimeter is not running. The screen shown is representative of our current configuration, but both sections of the screen allow for expansion.



Jun 13, 1990 (JD164) | AGSS Initialization | 10:32:25 Z

Primary Nav Source: GPS

Altimeter Offset(nsec): 12.3

Ground Pressure(mbars): 1001.3

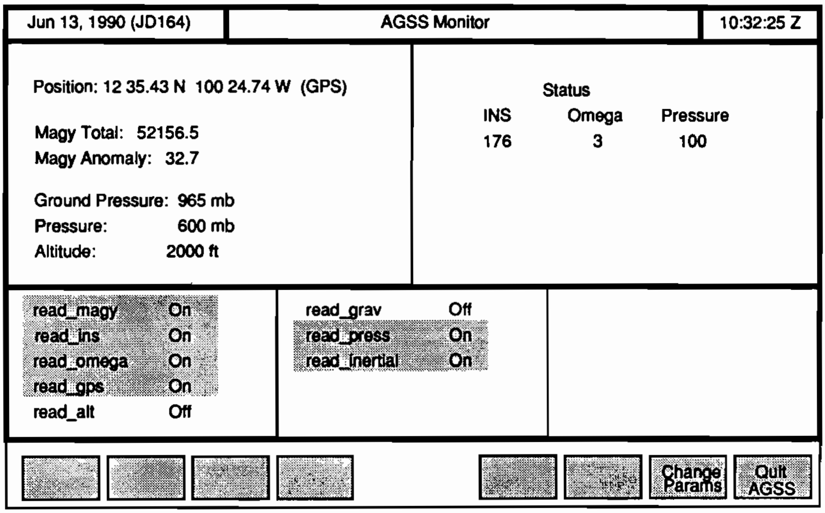
read_magy	Will run	read_grav	Will not run
read_ins	Will run	read_press	Will run
read_omega	Will run	read_inertial	Will run
read_gps	Will run		
read_alt	Will not run		

Figure 3
Typical Initialization Screen

Figure 4 shows a typical system monitor screen. This screen is displayed whenever the initialization screen is not displayed. The fields on this screen are not changeable by the operator. In fact when this screen is displayed the only keys which have any effect are the "Change Params" and "Quit AGSS" function keys as well as Ctrl-Z which is the kill key in our environment. In general the upper left portion of the screen shows the current value of measured data. The upper right part shows status of some of the data streams. The lower portion of the screen shows program status. If any of the running programs stops for some reason, its entry on the bottom of the screen will show "Not running" and will flash in order to get the operator's attention.

3.2 Implementation of the Interface.

When we decided how we wanted the interface to look, we thought that the obvious way to build it would be with the *curses* package of UNIX. We developed some preliminary mockups of the interface using *curses*, but were disappointed to find that the program was unreliable and extremely slow. Upon examining the terminal control sequences generated by *curses* we decided that we would never get the kind of system response we wished to have. We decided to develop a small library of routines to perform the functions which the HP2623A terminal supported. These include cursor movement, line drawing, function key usage, and highlighting of fields. The library which we developed, *crtlib*, is being made available through the swap tape for this conference. Table 1 summarizes the contents of *crtlib*.



Jun 13, 1990 (JD164) | AGSS Monitor | 10:32:25 Z

Position: 12 35.43 N 100 24.74 W (GPS)

Magy Total: 52156.5

Magy Anomaly: 32.7

Ground Pressure: 965 mb

Pressure: 600 mb

Altitude: 2000 ft

Status

INS	Omega	Pressure
176	3	100

read_magy	On
read_ins	On
read_omega	On
read_gps	On
read_alt	Off

read_grav	Off
read_press	On
read_inertial	On

Figure 4
Typical Monitor Screen

Many of the routines in *crtlib* simply send the appropriate escape sequence to the terminal. A few of the routines, however, require further explanation. The four routines for drawing lines use the line drawing set. They are called with one parameter which determines how many units (rows or columns) to move and in what direction (+ for up or right, - for down or left). Movement is always made relative to the current cursor location. We wanted it to be easy for the user to draw lines without worrying about whether they would join or cross properly. For instance, if a thick vertical line crosses a thin horizontal line the character at the intersection should be . In order to accomplish this the line drawing routines keep a memory image of what lines have been drawn on the screen. If a line passes through a cell which is already occupied, a table look-up is done based on the character encountered at that location and the type of line being drawn. The line drawing set is not quite complete, so a best approximation to the correct character has been entered in the table.

There are several things which must be done to use the function keys properly. There are actually several sets of function keys on the terminal. We are concerned only with the user function keys. First, the text to be displayed on the user keys and the value to be returned by the keys must be set up. This is done with the routine *set_fkey* which is called once for each function key. The first parameter for *set_fkey* is the number of the function key we want to set. There are twelve function keys on the 2623A, but only eight of them are normally displayed. We set all of them for completeness. The next two parameters are the upper and lower line of the text displayed on the key. These may be up to eight characters long. The last parameter is the value which will be transmitted when the key is pressed. We set them to an escape sequence which is unique to each key so that the *getkey* function can easily determine which key was pressed.

Name	Purpose
c_clear	Clear the screen
c_home	Home the cursor
c_moveto	Move cursor absolute
c_rmoveto	Move cursor relative
c_mlock	Lock screen
c_munlock	Unlock screen
c_hor_thick_line	Draw horizontal thick line
c_hor_thin_line	Draw horizontal thin line
c_ver_thick_line	Draw vertical thick line
c_ver_thin_line	Draw vertical thin line
set_fkey	Set the function keys
show_user_keys	Display the user function keys
show_mode_keys	Display the mode function keys
lock_fct_keys	Lock function key display
unlock_fct_keys	Unlock the function key display
set_XmitFnctn	Set transmit functions mode
clear_XmitFnctn	Clear transmit functions mode
ring_bell	Ring the terminal bell
getkey	Get a key stroke
change_field_style	Change field style
show_field	Display a field
edit	Alter value in text or numeric field

Table 1
Contents of *crtlib*

Next the user function keys must be displayed. This is done by *show_user_keys*. Then the user keys are locked onto the display so that the operator cannot change the display to any of the other sets of function keys. This is done by *lock_fct_keys*. Finally, *set_XmitFnctn* is called to cause the terminal to transmit the key contents instead of executing them locally. Whenever we exit from the program, we reset the terminal to its normal mode of displaying the mode function keys. Figure 5 shows the code which sets up the function keys for the display shown in Figure 4.

The routines *show_field* and *change_field_style* are higher level routines which operate upon a structure which describes a field on the screen. *Show_field* displays a given field, and *change_field_style* alters the field's appearance by setting the character mode settings for the field. This allows the programmer to easily change a field from, for instance, normal to inverse video. These routines perhaps should not be included in *crtlib* since they rely on a particular structure for the fields, but it is convenient to include them in this library. Figure 6 shows the structure which we use to define the fields on the screen.

The type of the field can be enumerated, integer, floating point, or text. If the type is enumerated, then *value* actually points to an integer between 0 and *no_choices* - 1. The *choice* array has entries which point to the text associated with each choice for the field. For instance, the highlighted field in Figure 3 is an enumerated field with *no_choices* equal to two. The *choice* array points to "Will run" and "Will not run". *Minv* and *maxv* are not used for enumerated fields. If the field is integer or floating point, then *value* points to the location which contains the field's value. *Minv* and *maxv* delimit the range of the value. *No_choices* and *choice* are not used. If the field is a text field, then *value* points to the text string. *Minv, maxv, no_choices,* and *choice* are not used. *Temp_value* is used to hold the current value of a field during field editing. When the value is accepted by the operator it is moved to *value* for use by the rest of the system.

```
/*      Set up the function keys      */
   set_fkey(  1, "       ", "        ", "\033p" );
   set_fkey(  2, "       ", "        ", "\033q" );
   set_fkey(  3, "       ", "        ", "\033r" );
   set_fkey(  4, "       ", "        ", "\033s" );
   set_fkey(  5, "       ", "        ", "\033t" );
   set_fkey(  6, "       ", "        ", "\033u" );
   set_fkey(  7, " Change", " Params ", "\033v" );
   set_fkey(  8, "  Quit ", "  AGSS  ", "\033w" );
   set_fkey(  9, " ", " ", "\033<" );
   set_fkey( 10, " ", " ", "\033=" );
   set_fkey( 11, " ", " ", "\033>" );
   set_fkey( 12, " ", " ", "\033?" );
   show_user_keys();
   lock_fct_keys();
   set_XmitFnctn();
```

Figure 5
Function Key Set-up

```
typedef struct { int column;      /* Position of the start of the   */
                 int row;         /* field                          */
                 int length;      /* Size of the field in characters*/
                 char *value;     /* Current value of field         */
                 char *temp_value;/* Temporary value for editing    */
                 int type;        /* Type of field                  */
                 float minv;      /* Minimum allowable value        */
                 float maxv;      /* Maximum allowable value        */
                 int no_choices;  /* Number of possible choices     */
                 char *choice[ MAX_CHOICES ];
                                  /* Array of choices for field     */
                                              } field_def;
```

Figure 6
Structure Defining a Screen Field

The final part of the operator interface is keyboard manipulation. When the initialization screen is active we want each key stroke to be processed immediately by the program. When the monitor screen is active we want only the function keys to respond. Processing key strokes immediately or responding to only certain keys is non-standard behavior for terminals. We strongly recommend that you read HP's *Asynchronous Serial Communications Programming Manual* before trying to do non-standard things with devices attached to the Mux card. Two things must be done in order to accomplish our goals. First, the mux port for the keyboard must be set to the non-canonical mode with echo off. Non-canonical mode means that the driver will not edit any key strokes. Note that the function keys and special keys (arrows, for instance) may return multiple character sequences. Figure 7 shows the code to do this. This code also saves the current terminal configuration so that it can be restored when the program stops. The time-out is set to one second to keep the program from hanging while waiting for operator response.

```
/* Save the current terminal configuration                  */
/* Set the terminal for non-canonical, no echo, 1 second time-out */
  if( (ret = ioctl( 0, TCGETA, &stermio ) ) < 0 ){
    perror( "params: TCGETA failed " );
    exit( -1 );
  }
  save_termio = stermio;
  stermio.c_lflag &= ~ICANON;
  stermio.c_lflag &= ~ECHO;
  stermio.c_cc[VMIN] = 0;
  stermio.c_cc[VTIME] = 10;
  if( (ret = ioctl( 0, TCSETA, &stermio ) ) < 0 ){
    perror( "params: TCSETA failed " );
    exit( -1 );
  }
```

Figure 7
Terminal configuration

Second, the keyboard must be read one character at a time. This is done by calling the *read* function of the C library with a count of one character. As noted some keys produce multi-character sequences which must be decoded. The routine *getkey* does this and translates the characters to a key code. Special action must be taken with the function keys since they send an escape sequence followed by a new line character. *Getkey* skips over the new line character.

The program *params* handles terminal I/O when the initialization screen is active. It uses *getkey* to get key strokes one at a time and act on them. Many of the allowable keys simply cause a movement of the active field on the screen. When the active field is a text or numeric field, values may be entered in the field. All of the keys on the keyboard either do something that makes sense for the title on the key, or they ring the terminal bell to indicate an illegal key. The routine *edit* is responsible for editing text and numeric fields. We have not included examples of the usage of *getkey* or *edit* here, but they can be found in the contributed *crtlib*.

4. Communication Among Programs.

Once the operator interface was working, our attention moved to the problem of inter-program communication. In the RTE version of the system, programs communicated through system common and class I/O. The equivalent facilities under HP-UX are shared memory and message passing. In addition, we wanted to insure that the terminal screen was not corrupted by programs which attempted to write to it while it was being updated by the monitor or initialization programs. This was done by using a combination of screen memory locking and semaphores. Each data acquisition program intercepts all interrupts and reports them to the monitor so that the monitor program can keep track of program status. Thus, we have used nearly all of the tricks described in the *Real-Time Programming Manual*. We found this manual to be well written and extremely useful in doing real-time programming. We strongly recommend that you read this manual (twice!) before doing any real-time programming.

4.1 Shared Memory.

Some of the acquisition programs require parameters which are set during initialization, and all of them supply data to the monitor program. This is done through shared memory since this is the fastest way to communicate. Data is available as soon as it is written into the shared memory. In our case only one program can modify any particular value in shared memory, so there is no need to worry about the integrity of the data. (If several programs can modify data in shared memory, then semaphores can be used to insure integrity.) Programs using data in shared memory simply assume that the data there is up to date. Figure 8 shows code for allocating shared memory. This is done by *agss* before starting any of the acquisition programs. When the acquisition programs are started they are passed a parameter which tells them where the shared memory is.

```
struct shmstruct{ struct prog_status progs[NO_PROGS];
                  int pressure_status;
                  int litton_status;
                  int radar_status;
                  int inertial_status;
                  int omega_status;
                  char pressure_type;
                  float pressure_p0;
                  float radar_offset;
                  char nav_source;
                  struct position current_position;
                  float current_magy_total;
                  float current_magy_anomaly;
                  float current_radar_alt;
                  float current_radar_delay;
                  short current_radar_tr;
                  short current_radar_mult;
                  float current_press_alt;
                  float current_pressure;
                  int mrmag_msgqid;
                };
/*   Get a shared memory segment and attach it and          */
/*   lock it into memory                                    */
   if( (shmid = shmget( IPC_PRIVATE, sizeof(struct shmstruct), 0644)) ==
                -1 ){
    perror( "(main): shmget failed" );
    exit( 1 );
  }
  if( (int)(shmptr = shmat( shmid, 0, 0 )) == -1 ){
    perror( "(main): shmat failed" );
    exit( 1 );
  }
  if( shmctl( shmid, SHM_LOCK, 0 ) == -1 ){
    perror( "(main): shmctl failed" );
    exit( 1 );
  }
```

Figure 8
Code for Allocating Shared Memory

The system function *shmget* allocates a block of shared memory of the requested size (in our case, the size of the structure *shmstruct*), and assigns access permissions to it. It also returns an identifier for the block. *Shmat* returns a pointer to the character at the beginning of the block. The call to *shmctl* locks the block into memory so that it cannot be swapped out. Note that we check each function for errors. For purposes of debugging the program, it is mandatory that all system routines which can return an error be checked for errors. In most cases errors indicate an unexpected problem with the system, and we chose to abort the program with an explanatory message to the operator.

Figure 9 shows one way in which the data in shared memory can be accessed. Remember that we allocated a block which was the size of a *shmstruct* structure, so everything in the block can be accessed as a member of a *shmstruct*. *Shptr* is made to be a pointer to the *shmstruct* which is located at the beginning of the shared memory block. We could, of course, access values in shared memory by using the -> operator applied to *shptr*. We prefer to use variables which are pointers to the various members of the *shmstruct*. The center portion of the code defines pointers to some of the values, and the last few lines of code show how they may be used to set values in shared memory. *Agss* passes the block identifier, *shmid*, of the shared memory block to each of the acquisition programs. Each of them may attach the block and access it using code similar to that in figures 8 and 9.

```
/*   Set up shared memory pointers   */
  shptr = (struct shmstruct *)shmptr;
  prstat = shptr->progs;
  press_type = &(shptr->pressure_type);
  press_p0 = &(shptr->pressure_p0);
  press_alt = &(shptr->current_press_alt);
  radar_offset = &(shptr->radar_offset);
  radar_delay = &(shptr->current_radar_delay);
  radar_tr = &(shptr->current_radar_tr);
  radar_mult = &(shptr->current_radar_mult);
  radar_alt = &(shptr->current_radar_alt);
  nav_source = &(shptr->nav_source);
  for( i = 0; i < NO_PROGS; i++ ){
    prstat[i] = progs[i];
  }
  *press_type = 0;
  *press_p0 = 1000.0;
  *radar_offset = 1234.567;
  *nav_source = 2;
```

Figure 9
Using Variables in Shared Memory

4.2 Message Passing.

Message passing in HP-UX roughly corresponds to program-to-program class I/O in RTE. The data acquisition programs use message passing to inform *agss* of any unusual situations. Messages are placed into a message queue. They can be taken out of the queue in the order in which they were entered. This mechanism allows several programs to place messages into the same queue, and indeed allows several programs to remove messages from the queue. We decided to use a different queue for each of the data acquisition programs in order to simplify the determination of who sent the message.

Figure 10 shows the code required to manage a message queue. *Msgget* returns a unique identifier for a message queue. We send this identifier as a run parameter to the acquisition program which is going to use it. *Msgrcv* is used to get the next message from the queue. Normally it will wait until a message is available, but the IPC_NOWAIT flag causes an immediate return to the calling program. If there is no message available then error is set to ENOMSG. Messages have a format which for the most part can be defined by the user. The message consists of a long integer which is the message type followed by any user defined structure. There is a system defined limit to the size of a message. The message type is assigned by the user and has no system defined meaning. The message type can be used to differentiate among different messages on the same queue, but we chose to use different queues for different message sources. Our messages always inform *agss* that an interrupt was intercepted by a data acquisition program which caused it to abort, so the structure of our messages consists of the message type followed by the integer interrupt number and a character string which describes the problem. *Msgrcv* puts the message into the structure pointed to by *msg*. The message queue is released by *msgctl* with a parameter of IPC_RMID.

```
struct msg_t{ long mtype;
              char reason;
              char text[80];
            };
/*    Get a message queue    */
  if( (msgqid = msgget( IPC_PRIVATE, 0644 )) == -1 ){
    perror( "(main): msgget failed" );
    exit( 1 );
  }
/*    Receive a message        */
  if( msgrcv( msgqid, &msg, msgsize, 0, IPC_NOWAIT ) == -1 ){
    if( errno == ENOMSG ){
       printf( "No message recieved\n" );
    }
    perror( "(main): msgrcv failed" );
    exit( 1 );
  }
  printf( "agss: %s aborted due to signal %d\n",
          prstat[i].name, msg.reason );
  printf( "      %s\n", msg.text );
/*     Release the message queue    */
  if( (msgctl( msgqid, IPC_RMID, 0 ) == -1) ){
    sprintf( message, "(main): msgctl( %d, IPC_RMID, 0 ) failed",
             msgqid );
    perror( message );
    exit( 1 );
  } else {
    printf( "agss: msgqid = %d removed \n",
               msgqid );
  }
```

Figure 10
Code to Manage a Message Queue

Figure 11 shows how the data acquisition program places a message into the queue. Any interrupt causes the acquisition program to enter an interrupt handling routine. The interrupt handler finds out which signal caused the interrupt and places the number of the signal and some explanatory text in the *msg* structure. *Msgsnd* puts the message into the desired queue. The *msgsize* parameter in the *msgsnd* call is the length of the message not including the message type.

```
/*     Send a message back to agss telling it that we are aborting and why.*/
/*     Then stop.                                                          */
  msgsize = sizeof(msg) - sizeof(long);
  sprintf( msg.text, "read_program: aborting because of signal %s",
          sig_names[sig] );
  msg.mtype = 1;
  msg.reason = sig;
  if( msgsnd( msqid, &msg, msgsize, 0 ) == -1 ){
    perror( "(handler): msgsnd failed" );
    exit( 1 );
  }
  exit( 1 );
```

Figure 11
Code to Send a Message

```
/*   Allocate semaphore for tty usage   */
#define TTY_SEM 0
  if( (semid = semget( IPC_PRIVATE, 1, IPC_CREAT | 0644 )) == -1 ){
    perror( "agss(main): semget failed" );
  }
/*   Set its initial value to one       */
  semarray[TTY_SEM] = 1;
  if( semctl( semid, 0, SETALL, semarray ) == -1 ){
    perror( "agss(main): semctl failed" );
    exit( 1 );
  }
/*   Get the semaphore, use the terminal, release the semaphore   */
    getsem( TTY_SEM );
    printf( "Now I can use the terminal\n" );
    releasesem( TTY_SEM );
/*   De-allocate the semaphores   */
  if( semctl( semid, 0 ,IPC_RMID, 0 ) == -1 ){
    perror( "agss: semctl IPC_RMID failed" );
  } else {
   printf( "agss: semaphore id released\n" );
  }
```

Figure 12
Code for Handling Semaphores

4.3 Semaphores.

We use a semaphore to control access to the terminal screen. The problem we had to overcome was that, once started, the programs in the system are independent and asynchronous. Each may write messages to the terminal, and we did not want them to be able to write to the terminal when the monitor or initializer are updating the menus on the screen. A semaphore is a system variable whose value can be altered or examined by cooperating programs. It is usually associated with a system resource (the terminal, for instance). When a program wishes to use the resource, it first looks at the value of the semaphore. If the value is greater than zero then the resource is presumed to be available. The program then decrements the semaphore (making the resource busy) and proceeds to use the resource. When it is done with the resource, it increments the semaphore. Assuming that no other program has decremented the semaphore it will become greater than zero, and the resource will again be available. HP-UX provides routines for allocating semaphores, changing their values, and checking their values. It also makes sure that only one program at a time can alter the semaphore. We decided to pass the identifier for the semaphore as a run parameter to each program which needed it. We then wrote library routines for getting (*getsem*) and releasing (*releasesem*) a semaphore. Figure 12 shows code for allocating a semaphore, using it to access the terminal, and de-allocating it. Note that the semaphore is initialized to one so that the terminal is available. When a program wants to use the terminal it gets the semaphore. If the semaphore is not available (i.e. its value is less than one), then the program will suspend until the semaphore becomes available. It will then lock it by decrementing it and use the terminal. When the program is through using the terminal it unlocks the semaphore. This scheme will only work if all programs which will use the resource agree to use the semaphore to gain access to the resource. Non-cooperating programs know nothing about the semaphore and will proceed to use the resource.

4.4 Interrupt handling.

We felt that it was important that the system operator be informed when any of the system's programs aborted. We wanted the program to write an informative message to the screen, and we wanted the monitor to do something to get the operator's attention. To accomplish this we have each of the acquisition programs handle all interrupts generated by the program. They then send a message to the monitor, and it in turn puts a flashing message on the screen.

An interrupt is caused by a signal. Signals can be generated by hardware conditions (e.g. floating point exception or device interrupt), by the system software (e.g. bad argument to a system call), or by the program itself. There are 32 different signals which can occur. Most of them have a system defined meaning, but a program can use any signal to mean anything it wishes. For instance, a program could generate signal SIGFPE (which normally means floating point exception) when the operator does not respond to a question. Of course, this might not be a good idea since it would then be difficult to tell when a floating point exception really occurred. Interrupts caused by any or all of the signals may be intercepted by the program by causing a particular routine to be executed when the signal occurs. There can be a separate routine for each signal or a single routine for all signals. We chose to intercept all signals (except SIGKILL and SIGSTOP which cannot be intercepted) and to handle all of them with a single interrupt routine. The response to all interrupts is to send a message to the monitor and to abort. This mechanism also makes it easy for the monitor to stop an acquisition program by sending a SIGUSR1 signal to it.

```
/*      Set up the interrupt handler to handle all signals    */
/*      except for SIGKILL and SIGSTOP                         */
  vec.sv_handler = handler;
  vec.sv_mask = 0;
  vec.sv_onstack = 0;
  for( i = 1; i <= 29; i++ ){
    if( (i != 9) && (i != 24) ){
      if( sigvector( i, &vec, 0 ) == -1 ){
        perror( "(main): sigvector failed" );
        exit( 1 );
      }
    }
  }
```

Figure 13
Code to Set Up Interrupt Handler

Figure 13 shows typical code required to intercept interrupts. The system routine *sigvector* causes the routine handler to be entered when a particular signal occurs. HP-UX only defines 29 signals, so only these are intercepted. Figure 14 shows a skeleton interrupt handler which prints a message about the cause and stops. Note here that the signal which caused the interrupt is passed as a parameter to the interrupt handling routine.

```
handler( sig )
  int sig;
{
/*     Ignore SIGINT, SIGTSTP, and SIGCONT      */
  if( (sig == SIGINT) || (sig == SIGTSTP) || (sig == SIGCONT) ){
    return;
  }
  printf( "read_litton: aborting because of signal %s",
           sig_names[sig] );
  exit( 1 );
}
```

Figure 14
Typical Interrupt Handler

5. Data Acquisition.

Having taken care of operator interface and inter-program communication problems, all we have to do is collect the data. All of our data gets to the computer in serial form through Mux cards or in parallel form through AFI cards. In a sense the AGSS system is not a good test for using HP-UX for real-time processing because all of its data is buffered in hardware. The serial data is buffered in the Mux card, and the parallel data is buffered in a custom built four channel parallel interface which holds up to 2048 samples per channel. This means that we can read blocks of data into the computer once or twice per second without worrying about losing data.

5.1 Serial data.

There are three usual problems in dealing with data over RS-232 lines. First, the two ends of the RS-232 lines must agree on pin assignment, handshaking method, and baud rate. Second, many devices supply data whenever they chose to instead of when the computer wants to read it. Last, many devices supply binary data instead of ASCII data. Our general rule of thumb for dealing with the first problem is that RS-232 connections will never work the first time they are tried. When this happens the first thing to verify is that the instrument is sending data on the line that the computer is receiving data on. Next check that the baud rates agree. Then check that the request-to-send and data-set-ready lines are properly connected or jumpered. We have found that the quickest way to do these checks is to connect an RS-232 break-out box to the lines and monitor the various lines with a scope. The second problem is handled by the Mux card which provides sufficient buffering for each channel on the card. The last problem is solved by careful configuration of the Mux card and driver.

Figure 15 shows the code for opening and configuring a serial port for our GPS receiver.First, the port is opened for reading. Then the current port configuration is read into the structure *stermio*. This structure is modified to have the desired characteristics. In this case we want to receive at 9600 baud, and we want to read 8 bit data with no parity and no driver processing of data (i.e. binary data). To accomplish this we set the input mode (*c_iflag*) to IGNPAR to ignore parity. The output mode (*c_oflag*) is set to zero although this is not significant for this device. The control mode (*c_cflag*) is set for 8 bit character size (CS8) and 9600 baud (B9600). The CREAD flag enables the receiver and the CLOCAL flag causes the driver to ignore the status lines on the port. Finally, the local mode (*c_lflag*) is set to zero which puts the port in non-canonical mode. This disables any driver processing of characters. Setting *c_cc*[VMIN] and *c_cc*[VTIME] to zero means that when a read is performed on this port any characters in the input buffer are returned to the program immediately. The value returned by *read* is the number of characters transferred which may be zero. This means that we must merely read the input buffer often enough to process the data being sent from the GPS receiver (at most a few times a second).

```
 int open_gps()
{
  int gps;
  int ret;
  struct termio stermio;
  if( (gps = open( GPS_PORT, O_RDONLY )) < 0 ){
    perror( "test_gps: Open of /dev/tty0pl failed" );
    exit( 1 );
  }
  if( (ret = ioctl( gps, TCGETA, &stermio )) < 0 ){
    perror( "test_gps: TCGETA failed" );
    close( gps );
    exit( 1 );
  }
  stermio.c_iflag = IGNPAR;
  stermio.c_oflag = 0;
  stermio.c_cflag = B9600 | CS8 | CREAD | CLOCAL;
  stermio.c_lflag = 0;
  stermio.c_cc[VMIN] = 0;
  stermio.c_cc[VTIME] = 0;
  if( (ret = ioctl(gps, TCSETAF, &stermio )) < 0 ){
    perror( "test_gps: TCSETAF failed" );
    close( gps );
    exit( 1 );
  }
  return( gps );
}
```

Figure 15
Code to Set Serial Port for Binary Input

5.2 Parallel data.

For the RTE version of the AGSS system we had built an interface for most of our parallel data which did some buffering and pre-processing of data before sending it to a Microcircuit Interface Card on our F-Series computers. We wished to use this same interface, but quickly discovered that there was no interface card for the 800 Series which was really equivalent. The closest card available is the Asynchronous FIFO Interface (AFI). It is an eight or sixteen bit parallel interface which has an on-board buffer of 66 words, but its handshaking method is quite different from the Microcircuit Interface card. We decided to redesign the handshaking circuitry of our special interface. The interface boards shipped with our system were the 27114A version which was the only one available at the time. We quickly discovered some major problems with this card. Initially, the most vexing problem was that if handshaking did not occur correctly (often the case for new designs), the hardware and software would hang in such a way that the program could not be terminated. Indeed, under some conditions the system would crash. A call to HP verified that there were some bugs in the driver, but we could overcome most of the problems by using the time-out feature on every call to the driver. This made it possible to check out our hardware design and to write software to deal with the card. In doing extensive tests of the interface, however, we found that we were getting errors at a rate of a few times per hour. The errors were of a nature that we could not blame on our software or hardware. The buffer that we were reading into was sometimes getting trashed by the driver. Another call to HP verified that this was a known bug, and again there was a bug fix. This time , however, the bug fix was hardware. There was now a B-version of the card and a new driver. The trouble was that the card was not available yet, and the driver would not be available until Revision 7.0 of HP-UX. We arranged with our local office to test our system with a pre-release version of the hardware and software and verified that the new board solved the problem. Through some fancy footwork by our local salesman we arranged to get two of the B-version boards and a pre-release copy of Revision 7.0 in time for our test flights. The moral of the story is to make sure that you buy the B-version of the card. It is a much more flexible card with many features not available on the A-version. The driver fully supports all modes of the board and has been found to be reliable.

Handshaking on the AFI card is done by two lines, PCNTL and PFLG. When reading data into the card (i.e. the user performs a read), the card first asserts the PCNTL line. The external device should respond by placing its data on the data lines and then pulsing the PFLG line. The card then de-asserts PCNTL and reads the data. Since the card has on-board buffering, it will then proceed to assert PCNTL again. This will continue until the buffer is full, the external device stops sending data, or the card is set to output mode. Because of this action it is useful for the program to know how many data values to read from the external device. In our case we can ask the external device how many data points it has available and then read that many values.

```
/*     Open the gpio device to do I/O
  if( (gpio = open( "/dev/gpio0", O_RDWR )) == -1 )(
    perror( "(main): open of gpio failed" );
    exit( 1 );
  )
/*    Set timeout to 10 seconds for reset      */
/*    Reset the gpio interface card     */
/*    Set timeout to 5 sec    */
  io_timeout_ctl( gpio, 10000000L );
  gpio_ctl.type = GPIO_RESET;
  gpio_ctl.arg[0] = HW_CLR;
  if( ioctl( gpio, IO_CONTROL, &gpio_ctl ) == -1)(
    perror( "(reset_gpio): ioctl reset failed" );
    exit( 1 );
  )
  io_timeout_ctl( gpio, 5000000L );
/*   Set to trigger on trailing edge (low to high) of PFLG     */
  gpio_status.arg[0] = GPIO_GET_CONFIG;
  if( ioctl( gpio, IO_STATUS, &gpio_status ) == -1 )(
    perror( "ioctl status failed" );
    exit( 1 );
  )
  gpio_ctl.type = GPIO_SET_CONFIG;
  gpio_ctl.arg[0] = gpio_status.arg[0] | (EDGE_LOGIC_SENSE);
  if( ioctl( gpio, IO_CONTROL, &gpio_ctl ) == -1)(
    perror( "(reset_gpio): ioctl IO_CONTROL failed" );
    exit( 1 );
  )
/*     Read a record      */
    while( (nread = read( gpio, (char *)inbuf, n*4 ) == -1 );
```

Figure 16
Code for Handling AFI Card

Figure 16 shows code necessary to open, configure, and read, a device connected to an AFI card. The device is opened with a simple open call to the device file for the card you wish to use. Remember to immediately set the timeout for the card. We have found that the time out should be set to at least 5 seconds at all times. During reset it should be set to 10 seconds.

Control of the card is done through the *ioctl* function. The first parameter of this function specifies the device to be controlled. The second parameter specifies the type of the function to perform (IO_CONTROL or IO_STATUS). The third parameter is a structure which specifies the function to be performed and supplies any data needed by that function. The sample code shows how to do a full reset on the card. This resets all flags on the card to a known state and clears the input and output buffers on the card. Then we set the card to operate on the trailing edge of the PFLG handshaking pulse. Notice that to do this we first read the card status (using IO_STATUS). Then we alter those configuration bits which need to changed and write the configuration back out to the card. The function read is used to get data from the card. *Inbuf* is an array of long integers and n is the number of 32 bit words to read. If the read fails due to a time-out the return value is -1, so we loop until we actually get some data. It is good idea to read the interface manual carefully before using the AFI card. In addition, it may be useful to read the manual page for gpio.

6. Results of Flights.

The AGSS system is being used successfully by researchers at NRL and other institutions, including American and foreign universities, to collect geophysical data all around the world. The various field experiments collect sets of data from the different sensors depending on what is being investigated, but at a minimum, at least one sensor and navigation source. Moreover, we expect to add new sensors and measurements in the future. This will not be difficult due to the modular design of the software.

Because we work with so many different institutions and people, the project operators change with each expedition. Since December, 1989, the AGSS system as implemented on the HP-9000/835 has been tested in three different experiments. In December, 1989 the system was used to collect inertial and dynamic GPS navigation data for complementary processing and comparison. In January, 1990 the system was used to collect magnetics data over the Chile Rise operating out of Puerto Montt, Chile. Currently the system is being used in the Indian Ocean to collect magnetics data to characterize tectonic plate motions in that region. Each of these experiments has been run with different personnel and the last two without personnel experienced with UNIX. Comments from the project operators have been favorable: the screen menus make it virtually a turn-key system and very easy to use.

More important than extensibility or ease of operation is reliability of data collection. Flight time is expensive and fuel and personnel costs are high. It is critical to monitor the equipment and record the sensor measurements. The HP-9000/835 has performed well in the P-3 environment, but problems do occur with the measurement devices. When the component programs that read data stop in error, the operator is notified at the console. This makes it easy to monitor all the hardware, which is dispersed throughout the cabin, from one location and makes data collection more reliable while reducing the numbers of observers required.

The *agss* program has been a success for several reasons. It is easy for people to run the system to collect data. It is a versatile system that handles many combinations of sensors and instruments in addition to being extensible to include new measurement sensors. The data is not only reliably recorded, but the operators are warned of instrument failure. Finally, using an industry standard UNIX system allows us to more easily exchange data and programs with other research organizations.

7. Conclusion.

Based on our experience in porting the AGSS system from RTE to HP-UX we can make several recommendations. First, read all of the appropriate manuals before starting to write code. Manuals to pay particular attention to are the *Real Time Programming Manual* and the *Asynchronous Serial Communications Programming Manual*. Both manuals are well written and contain far more information than we could provide here. Second, if you need to develop a menu driven system with an alphanumeric terminal, we believe that you should not try to use *curses* but should write your own terminal handling routines. The library that we wrote is a good starting point for such a library. You need to keep in mind that it is specialized for the 2623A terminal and that it only implements those functions which we needed for our application.

Our third recommendation is that you learn to use the UNIX utilities. In particular, *make* is extremely useful in making sure that all parts of a multi-program system get updated when they need to be. As an example, a change in a library routine should cause all programs which use that library to be rebuilt. *Grep* is very useful in finding occurrences of variables in programs, and learning the basics of using regular expressions is important for effective use of the *vi* editor.

Finally, we recommend that you convert systems to HP-UX native mode as soon as possible. Even though this may require you to rewrite your system in C, we believe that the benefits far outweigh the time consumed in doing this. The AGSS system contains about 10000 lines of C code. The port was done over about a six month period. This included the time needed to change all code from FORTRAN to C as well as to develop the new user interface. In a sense the AGSS system is not a good test of porting real-time systems to HP-UX because nearly all of the data is buffered in hardware. However, our experience has convinced us that we can base our future real-time systems on the HP 9000/835 computer using HP-UX.

WPMS- A Centralized Performance Monitoring
System for Engineering Workstations

Juan Luis Arroyo and Mario Jauvin
Bell Northern Research
Station C
PO BOX 3511
Ottawa, Ontario
Canada
Phone: 613 7637944

Workstation Performance Monitoring System

Bell Northern Research (BNR)

Bell-Northern Research is the Research and Development (R&D) arm of Northern Telecom, a global communications company whose primary products are Central Office Switches for the Bell Operating Companies and the long distance carriers, Private Branch Exchange (PBX) equipment for the office market and transmission products, such as our Fiberworld family.

There are more than 6000 employees in BNR. Apart from Ottawa we have two major centres in Richardson Texas and Raleigh North Carolina. There are three smaller locations in Montreal, Atlanta and Maidenhead in the United Kingdom.

The majority of employees are engaged in Software and Hardware design for the products mentioned above. The remainder are engaged in support activities such as quality control and testing, as well as the traditional administrative functions.

As can be seen, the business we are engaged in is computing intensive, and our investment in Information Technology is key to our ability to succeeed in an extremely competitive market place.

The Computing Environment

The current computing environment at BNR consists of three primary platforms:

o The mainframe has been our primary design environment in the 80's. It was used for hardware design, programming and administrative and support applications.

o The capture, presentation and selling of ideas is a key aspect of the R&D business. The Macintosh PC gained rapid acceptance in BNR when it was introduced in the mid 80's, because of its exceptional ability to do presentation graphics, coupled with its ease of use. We have as well, several hundred PC's, plus numerous other systems such as VAX used in special purpose applications, such as testing.

o In 1986 we began to explore the use of UNIX workstations, specifically for the hardware design side of our business. By 1988, we decided to migrate our hardware design environment to the workstation, and chose the Apollo workstation, using ethernet Lans and Apollo's version of UNIX BSD. Also in 1988, we began to explore the use of workstations for our software design environment, and in 1989 we decided to move that environment to workstations as well.

- By the end of 1990, we expect to have 3,000 workstations deployed across the corporation. It is a multi-vendor environment with HP and SUN motorola based workstations being used for software development and the Apollo and SUN Sparc workstations used for hardware design.

Capacity Planning and Performance Management:

With 9 large mainframes in the Ottawa area and 5 regional processors providing computing capacity essential to our R&D activities, we have come to appreciate the value of the capacity planning and performance management function. Over the years, we have developed numerous tools and gained invaluable experience in understanding and controlling the factors that determine performance on the mainframe. A comprehensive reporting system which includes exception reports, real-time reports, daily reports and history reports allows us to forecast capacity requirements and configure our mainframes based on factual data. Our reporting system also allows us to prevent most performance problems through proactive tuning.

The Capacity Planning and Performance Management function applied to engineering workstations is as necessary and important as in the case of the mainframes. For example a decision to add 8 Mbytes of memory to 1,000 of our workstations represent an investment of US 2.5M, a decision to share workstations using X-terminals saved us US$ 10 M in 1990 capital. The decision to use dataless clusters saved us US$ 2 M. We could not have made these decisions in the absence of factual performance measurements. Dataless clusters are groups of approximately 30 HP9000/360 workstations with 161 Megabyte disks sharing a file server which contains the operating system and user data. Local disks are used only for page/swap activity.

Regarding Performance Management: whereas in the mainframe we had to deal with 14 machines whose performance data was readily available to us so that we could anticipate performance problems and verify and resolve users complaints, in the workstation world we have to deal with about two thousand machines and initially had no historical data to analyze when faced with performance problems.

The Workstation Performance Monitoring System- Functionality

Our early experience solving performance problems in the workstation environment led us to conclude that the most cost effective way to implement a comprehensive capacity planning and performance management function required a system to collect, reduce and store performance measurements for all our UNIX workstations, WPMS is such a system.

WPMS is a client/server based system, where the clients are the workstations being monitored and the server is the central node to which the performance data collected on the client workstations is sent at fixed time intervals.

WPMS has five main components, data collection facility, data transport facility, data reduction facility , performance database and online reporting system. These components will be discussed in greater detail in another section of this paper.

WPMS provides our performance analysts with factual information to support their capacity planning and performance management function, in the form of on-line reports tailored to their requirements. The main reports are as follows: (samples are included)

Daily exception report:

Workstations get placed on this report when their key performance indicators deviate from thresholds associated with adequate performance. The main use of this report is in the detection and elimination of runaway processes from our workstations. This process is fully automated to the point where a piece of electronic mail is sent to the owner of the runaway process without any human intervation with instructions on how to get rid of it. Some other applications of this report have not been so succesful due to factors which we will explain in the section "Drawbacks and shortcomings".

Daily and Month-to-date summary report.

Daily and monthly performance data for every workstation are found in these reports. They are useful to carry out comparative studies of workstations running different applications and performing different functions in the organization and to provide managers with a performance summary for their workstations.

Workstation history report

It contains all daily performance metrics for a given workstation since the time WPMS started running on it. It is extremely useful in tracking down performance problems for individual workstations.

Detailed performance reports

This is a trouble shooting report that helps the analyst in determining what hardware resources were bottlenecked when users were experiencing performance problems.

The Workstation Performance Monitoring System- Usefulness:

WPMS has proven useful in the following areas:

Capacity Planning

- o Support of upgrade decisions- Resource utilization trends are used to justify or discourage hardware upgrade decisions.

- o Hardware configuration- helped us determine the amount of memory required on the host workstation to support X-terminals in our software development environment; WPMS measurements played a key role in our decision to use dataless systems (clusters of approximately 30 HP9000/360 workstations with 161M disks sharing a file server which contains the operating system and user data. Local disks are used only for page/swap activity)

- o Computer growth analysis-WPMS performance data summary reports are distributed monthly to BNR managers to allow them to assess the computer resource consumption of their groups

- Runaway processes-On a typical day we find runaway users on 10% of our HP workstations. Most of them are are caused by the IBM 3278 terminal emulation processes (tn3270). WPMS has allowed us to optimize the detection and correction process.
- Bottleneck Analysis-Often WPMS has proven to be an invaluable tool in determining the cause of a performance problem. As an example, let's say a user calls us saying that he has been experiencing poor response time on his workstation in the last few days. We would look at the detailed performance reports searching for changes in the performance metrics to determine the cause of the problem.
- Configuration parameters-WPMS performance metrics can be used to estimate swap space and I/O buffer size requirements. We also use it to monitor the effect of running X-terminals off the workstations.
- Tracking Measurements-The availability of history data for every workstation allows us to assess the impact of a change in the environment or the introduction of new software on workstation performance.
- Learning tool-Having a sound regular set of reports that provides insights into the system has contributed to our learning process by allowing us to observe the impact on performance metrics of changes to the environment.
- Productivity improvements through automation. Our performace analysts do not have to spend time setting up performance mesurement tools and reducing the data on a case by case basis. Most of the data they need to do their analysis is already captured. They do not have to reproduce a problem either because the performance measurements already exist.

Drawbacks and shortcomings

Development of WPMS was not without difficulties, and the first version of our software did not satisfy all our expectations. We are planning to release an enhanced version in September.

WPMS development started in February 1989. At the time, our knowledge of workstation performance was very limited since our department had just received its first HP/360 running HP-UX 6.2 in late January. We made the decision to use vendor supplied tools as the data extractor element in our system and chose "Monitor" for HP workstations and VMSTAT for both Apollo and SUN's. The purpose was to keep development and maintenance costs down by not having to write our own performance data extraction software, we assumed vendors would enhance their software with each release of the operating system and we would include these new improved versions on our evolving system. As it turned out, the HP monitor did not provide all the information we were looking for: The CPU utilization did not include some components such as CPU consumed when handling interrupts, CPU consumed by niced processes and CPU consumed by the system in kernel mode; The I/O rate supplied by monitor did not measure the physical I/O rate to disk; the device utilization was not provided for SCS1 disks etc. The HP monitor is a good tool to investigate some types of performance problems but did not fit all our data collection requirements.

SUN's implementation of VMSTAT was not adequate for our purpose either because it did not provide an average of the performance metrics for the collection interval for all variables. Some of the measurements were averages for only the last 5-seconds sampling interval.

Our original objective was to be able to define performance variable thresholds for good performance so that we could automate the process of evaluating service levels. The inadequate capabilities of the existing performance monitors prevented us from doing this.

This led us to develop our own performance data extraction tool for both HP an SUN workstations using the vmmeter and vmtotal data structures in dev/kmem and the files dev/mem and /dev/swap.

There are two very important gaps in our performance metrics and we encourage vendors to help us close them. The first concerns user perception metrics such as response times and expansion factors. The second is a set of metrics on the X-server which has a direct impact on the performance of interactive commands.

System overview

WPMS is a client/server based system, with the clients being the workstations that we monitor and the server being a central node(s) where the data is sent and processed.

WPMS has five components, data collection facility, data transport facility, data reduction facility, performance database and report generator. The first two subsystems run on the workstation whereas the last three are mainframe based. Each component has been designed such that it can be modified or replaced without greatly impacting the other components. This modular approach allows us to install upgrades (eg. a new data collection facility quickly and easily. Simplicity, low overhead and growth potential were our main design objectives.

Data collection facility

The data collection facility resides on all installed workstations (clients). The first component is made up of the monitoring tools available with each workstation platform (proprietary tools or system commands). We are currently using MONITOR for HP's and VMSTAT for the Apollos, The SUN version is not yet in operation. The monitoring tool wakes up at given intervals (currently 10 minutes) and produces a line containing average performance metrics for the interval. A shell script collects header data (workstation model, name,time,etc) and combine these with the monitor data to produce one output record which summarizes workstation performance measures. A buffering scheme has been set up which allows several output records to be buffered and sent as one packet to the server.

The current vendor supplied monitoring tools provide us with only a subset of the total performance data we require. We are currently working on our own monitoring tools to obtain some of the additional data we need.

Transport facility

The transport facility consists of two parts, one part resides on the client workstations and sends data to a server, the second resides on the server and receives data from the clients.

The "send" program, that is installed on all client workstations, is a "C" program that receives the performance data from the monitoring programs as "standard input (stdin)". The program buffers the data until we have a predetermined number of observations and then sends it to the server using UDP/IP (User Datagram Protocol). Using UDP/IP does not guarantee delivery of the data packet, however it was chosen because of its low overhead cost.

The "receive" program that runs on the server, receives all UDP packets from the clients and writes the data to a disk file. A new raw data file is created each day with the date as part of the file name. Currently this program runs on a mainframe and receives packets from all workstations via KNET's and K200's which implement TCP/IP servers on the mainframe.

Data Reduction Facility

We are using SAS programs to reduce and process the raw data file and produce the desired performance information (eg. average device utilization, Input/output rates, page rates, device and system queueing information). The raw data file is transformed and appended to SAS dataset(s). Other functions such as checking for lost packets, "degrouping" packets, etc. are also performed.

SAS provides us with a flexible tool not only for reducing the data but also for doing performance analysis. Also, SAS is being used to merge in data from our Telecommunication Operations System (TOS) (user name, department and physical location) and produce a summary record for each node.

Performance Database

This database is resident on the mainframe and contains the latest reports for each workstation. Old reports are archived to tape. Raw monitor data is also available for a shorter period On-line and then is also archived to tape.

Report Generator

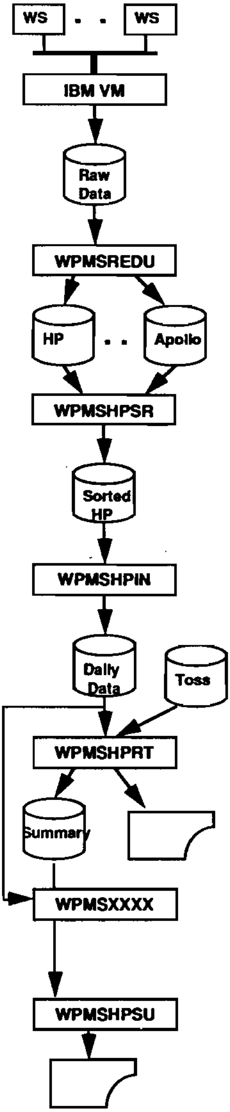
The report generator is a SAS program that extracts performance measurements from the SAS dataset and summarizes them into various reports. Reports can be produced in various formats depending upon requirements (eg. by date, node, hour, workstation model, location, department, etc). Exception reports that flag potential problems are also produced.

WPMS cost

Development of WPMS took 6 person-months. The central server runs on an IBM 3081K under VM/CMS and uses about 0.2% of its capacity and 800 IBM 3380 cylinders (560

Mbytes) of disk storage. WPMS monitors performance on 800 HP 360 and 300 Apollo 4500 workstations, the SUN version is currently being developed. WPMS uses up 0.14% of our LAN capacity and the three processes running on each client workstation consume 0.03 % of their CPU capacity with virtual memory requirements of 600 Kbytes on each client workstation.

Workstation Performance Monitoring System (WPMS)
Overview of System Flow



Over 1,100 workstations in Canada and US send packetized records to a mainframe server

Records are 'ungrouped' and converted from EBCIDIC to ASCII

Each day more than 30 Mb of performance data is captured

SAS is invoked to interpret the data, seperating it into one SAS dataset for each vendor of WS

These SAS datasets are sorted by workstation node and datetime.

Calculations are done to convert some data from counters to rates, also detection of workstation boot and missing data is done here.

A summary dataset containing one observation per WS per day is created, WS descriptive data from TOSS is merged in. Daily reports by shift and by WS are created.

Several reports are produced here:

- Exception report (ie WS where CPU > 99%)
- Frequency distribution of CPU
- WS install base by model and location

All daily files for a month are concatenated to produce month-to-date reports by node, workgroup etc.

WPMS EXCEPTION REPORTING

Exceptions for HP Workstations

```
BCARH106 BRUNET       7V50   Check Paging, PAGEREQ=  49.4 STD=  14.2
BCARH11                      May be runaway, TOTCPU= 96 STDCPU= 14
BCARH138 GALVIN       4Y44   Check Disk Activity, DISKIO  =  45
BCARH139 NEWCOMBE     7X51   May be runaway, TOTCPU= 99 STDCPU=  0
BCARH215 REDDING      7Z32   Check Swap Activity, MAXSWAP= 92 SWAP= 69
                               STDSWAP= 16
BCARH219                     May be runaway, TOTCPU= 99 STDCPU=  0
BCARH33                      Check Swap Activity, MAXSWAP= 91 SWAP= 71
                               STDSWAP= 20
BCARH347 LEIBA        7N42   May be runaway, TOTCPU= 99 STDCPU=  0
BCARH349 CHUNGPHAISAN 7Y11   May be runaway, TOTCPU= 99 STDCPU=  0
BCARH407 DANKO        7B31   Check Swap Activity, MAXSWAP= 93 SWAP= 90
                               STDSWAP=  1
BCARH418 YANG         7B31   May be runaway, TOTCPU= 99 STDCPU=  0
BCARH420 LUI          7Z11   Check Paging, PAGEREQ=  44.8 STD=  10.0
BCARH471 GOLLER       4F01   Lan Packet rate excessive LANPACIO =  249
BCARH498 AMALU        7X44   May be runaway, TOTCPU= 90 STDCPU= 27
BCARH512 BROWN        7X71   May be runaway, TOTCPU= 99 STDCPU=  0
BCARH514 ZAHARYCHUK   7N10   Lan Packet rate excessive LANPACIO =  160
BCARH543 POPOFF       2B71   May be runaway, TOTCPU= 99 STDCPU=  0
BCARH549 PERSONNA     7Z32   May be runaway, TOTCPU= 99 STDCPU=  0
BCARH576 KENT         7H25   May be runaway, TOTCPU= 91 STDCPU= 27
BCARH66  CARLYLE      7N11   Check Paging, PAGEREQ=  26.6 STD=  36.9
BCARH80  SINGH        7B52   May be runaway, TOTCPU= 99 STDCPU=  0
BMERH113 CAVAN        8R41   May be runaway, TOTCPU= 99 STDCPU=  0
BMERH131 KILNER       7D73   May be runaway, TOTCPU= 99 STDCPU=  0
BMERH217 SUTHERLAND   7Z31   Lan Packet rate excessive LANPACIO =  114
BMERH237 HATZ         7B61   Check Paging, PAGEREQ=  38.4 STD=   6.8
BMERH253 PORECHA      8L12   May be runaway, TOTCPU= 99 STDCPU=  0
BMERH548 SAINI        7I21   May be runaway, TOTCPU= 99 STDCPU=  0
BMERH556 HUMPHREYS    7I00   Lan Packet rate excessive LANPACIO =  128
BMERH644 DEERY        7L22   May be runaway, TOTCPU= 99 STDCPU=  0
BMERH652 CUNNINGHAM   7M10   May be runaway, TOTCPU= 99 STDCPU=  0
BRCHH139                     Lan Packet rate excessive LANPACIO =  133
BRCHH139                     Check Disk Activity, DISKIO  =  42
BRCHH146                     Lan Packet rate excessive LANPACIO =  133
BRCHH146                     Check Disk Activity, DISKIO  =  41
BRCHH147                     Lan Packet rate excessive LANPACIO =  130
BRCHH90                      Lan Packet rate excessive LANPACIO =  105
BRTPH140                     May be runaway, TOTCPU= 99 STDCPU=  0
BRTPH156                     Check Paging, PAGEREQ=  30.3 STD=   1.1
BRTPH156                     Check Swap Activity, MAXSWAP= 92 SWAP= 91
                               STDSWAP=  1
BRTPH160                     May be runaway, TOTCPU= 99 STDCPU=  0
BRTPH64                      Check Swap Activity, MAXSWAP= 91 SWAP= 65
                               STDSWAP= 23
BWDLH31  MCAULIFFE    1U13   May be runaway, TOTCPU= 99 STDCPU=  0
```

```
                                                        DAILY SUMMARY REPORT FOR HP WORKSTATIONS

        NODE     USER   USER                  WS      %USER %SYS %TOT  MAX %WS %MON   %   MAX   %   MAX  PAC   ID/ DISK/ CALLS PAGES
 DATE   NAME     DEPT   NAME                MODEL      CPU  CPU  CPU  CPU ACT COV   MEM  MEM SWAP SWAP /SEC  SEC BUS   /SEC  /SEC
 ----  --------  ----  ----------         ---------  ----- ---- ----  --- --- ----  ---  --- ---- ---- ----  --- ----  ----- ----

27APR90 BCARH124 9D35 AZAD                9000/360       0    1    1    1 100   21   57   57   14   14   26  0.6    0    22  0.0
27APR90 BMERH109 8R41 CAVAN               9000/360       1    1    2   15  23  100   75   95   34   52    2  0.5    0    43  0.5
27APR90 BMERH111 8R41 CAVAN               9000/360      45   40   86   99  98  100   58   95   28   67    3  0.8    0  2516  0.9
27APR90 BMERH112 8R41 CAVAN               9000/360       1    1    1   12  23  100   77   95   40   53    2  0.5    0    67  0.5
27APR90 BMERH113 8R41 CAVAN               9000/360       1    1    3   18  37  100   77  100   41   59    3  0.5    0    75  0.9
27APR90 BMERH115 8R41 CAVAN               9000/360      41   26   67   99  72   81   49   64   21   23    2  0.5    0  1540  0.1
27APR90 BMERH117 8R41 CAVAN               9000/360       1    1    2   23  24  100   74   96   36   55    2  0.4    0    68  0.4
27APR90 BMERH17  8R41 REID                9000/360       0    0    0    4  25   88   73   95   28   38    2  0.3    0    11  0.1
27APR90 BMERH18  8R41 REID                9000/360      19   13   31   99  70   96   71   99   29   39    2  0.5    0   704  0.1
27APR90 BMERH19  8R41 REID                9000/360       0    1    1   13  18   92   61   95   23   36    2  0.4    0    16  0.1
27APR90 BMERH202 8R41 CAVAN               9000/360      11   88   99   99 100  100   58   95   28   57    2  0.6    0  6237  0.8
27APR90 BMERH207 8R41 REID                9000/360       0    0    1   15  16   92   52   95   20   38    2  0.4    0    14  0.1
27APR90 BMERH208 8R41 REID                9000/360       0    0    1   12  15   96   82   94   32   37    2  0.3    0    12  0.1
27APR90 BMERH21  8R41 BEZANSON            9000/350       0    1    1    8 100  100   24   25    5    5    3  1.7    1    33  0.3
27APR90 BMERH22  8R41 REID                9000/350      60   38   97   99 100  100   66   74   12   14    7 16.4    3  4752  0.2
27APR90 BMERH24  8R41 REID                9000/360       1    1    1    9  36   87   87   95   40   45    3  1.5    0    22  0.2
27APR90 BMERH25  8R41 BROWN               9000/360       0    0    0    0   0  100   36   36   15   15    2  0.1    0     1  0.0
```

```
                                             SUMMARY DF PRIME TIME WORKSTATION HISTORY
                                               FOR HEWLETT PACKARD       WORKSTATIONS

         NODE    USER   USER            WS      %USER %SYS %TOT  MAX  %WS %MON    %  MAX    %  MAX  PAC  IO/ DISK/ CALLS PAGES
   DATE  NAME    DEPT   NAME          MODEL       CPU  CPU  CPU  CPU  ACT  COV  MEM  MEM SWAP SWAP /SEC  SEC   BUS  /SEC  /SEC
   ---- -------- ---- ----------     ---------  ----- ---- ----  ---  --- ----  ---  --- ---- ---- ----  ---  ---- -----  ----

02APR90 BWDLH58  4Y34 JAUVIN         9000/380       0    2    2   28   20  100   50   70   30   39    4  3.8     0    34   0.6
03APR90 BWDLH58  4Y34 JAUVIN         9000/360       2    5    7   45   21  100   78   95   41   48    5  4.9     0    89   0.8
04APR90 BWDLH58  4Y34 JAUVIN         9000/360       3   12   15   99   21  100   90   96   54   61    5  5.0     0   610   0.8
05APR90 BWDLH56  4Y34 JAUVIN         9000/360       3    5    8   48   21  100   79   97   50   70    5  4.4     9    91   0.9
06APR90 BWDLH58  4Y34 JAUVIN         9000/360       2    5    7   42   19   97   77   96   41   53    6  4.5     0    89   0.7
09APR90 BWDLH58  4Y34 JAUVIN         9000/360       1    3    4   42   21  100   74   91   39   46    4  4.0     0    68   0.6
10APR90 BWDLH58  4Y34 JAUVIN         9000/360       1    4    5   53   21  100   81   94   41   47    4  4.0     0    74   0.6
12APR90 BWDLH58  4Y34 JAUVIN         9000/380       2    5    7   36   21  100   77   97   46   56    5  4.4     0    89   0.6
13APR90 BWDLH58  4Y34 JAUVIN         9000/360       0    3    3   19   20  100   51   56   35   38    4  4.5     0    41   0.6
16APR90 BWDLH58  4Y34 JAUVIN         9000/380       1    4    5   34   20  100   81   96   47   55    4  4.8     0    80   0.7
17APR90 BWDLH58  4Y34 JAUVIN         9000/360       2    5    7   38   20  100   82   97   49   61    8  5.4     0    96   0.8
18APR90 BWDLH58  4Y34 JAUVIN         9000/360       3    8   11   54   20  100   89   97   61   67   16  6.4     0   148   1.2
19APR90 BWDLH58  4Y34 JAUVIN         9000/360       3   12   16   65   20  100   85   97   62   73   20  9.1     0   167   1.6
20APR90 BWDLH58  4Y34 JAUVIN         9000/360       4   13   16   48   20  100   87   97   68   88   18  8.5     0   163   1.8
23APR90 BWDLH58  4Y34 JAUVIN         9000/360       2   11   13   44   21  100   84   97   58   74   15  8.0     0   127   1.6
24APR90 BWDLH58  4Y34 JAUVIN         9000/360       1   10   12   39   20  100   84   97   58   79   19  7.9     0   128   1.7
25APR90 BWDLH58  4Y34 JAUVIN         9000/360       3   12   15   99   20  100   77   99   57   97   13  8.9     0   160   2.0
28APR90 BWDLH58  4Y34 JAUVIN         9000/360       7   13   21   98   20  100   82   98   59   97   15  8.4     0   157   3.8
27APR90 BWDLH58  4Y34 JAUVIN         9000/360       5   13   19   71   22  100   84  100   62   96   17  9.3     0   195   3.4
```

```
                                                                DETAILED PERFORMANCE REPORT
NODE: BWDLH58                                                                                       14:49 MONDAY, APRIL 30, 1990    7

                     MONITOR  NODE        WS        UNIX     %USER %SYS %TOT MAINFR.  %    %   LANPAC OISK  %DISK SYSCALLS PAGES
 DATE      TIME      INTERVAL NAME       MODEL     VERSION    CPU   CPU  CPU   MIPS  MEM  SWAP ID/SEC IO/SEC BUSY   /SEC    /SEC
 ----      ----      -------- --------  ---------  ---------  ----- ---- ----  ------- --- ---- ----- ----  ----  --------  ----

27APR90 10:25:47     120    BWDLH58  9000/360  HP-UX 6.5   13    24   37    1.0    93   62   20.8   17.4    0    257.4     3.5
27APR90 10:27:47     120    BWDLH58  9000/360  HP-UX 6.5    4    14   18    0.5    93   63   12.1    8.2    0    170.8     1.4
27APR90 10:29:47     120    BWDLH58  9000/360  HP-UX 6.5   10    18   28    0.8    92   62   19.6   16.3    0    203.3     3.3
27APR90 10:31:47     120    BWDLH58  9000/360  HP-UX 6.5   11    22   33    0.9    96   63   16.9    9.6    0    379.4     2.4
27APR90 10:33:47     120    BWDLH58  9000/360  HP-UX 6.5   18    26   44    1.2    92   62   10.4    9.2    0    592.5     1.4
27APR90 10:35:47     120    BWDLH58  9000/360  HP-UX 6.5   26    33   59    1.6    96   64   19.0   10.3    0    655.1     2.1
27APR90 10:37:47     120    BWDLH58  9000/360  HP-UX 6.5   22    32   54    1.4    92   63   17.0    9.6    0    612.4     2.0
27APR90 10:39:47     120    BWDLH58  9000/360  HP-UX 6.5    5    18   23    0.6    87   61   19.4   11.8    0    187.6     2.2
27APR90 10:41:47     120    BWDLH58  9000/360  HP-UX 6.5    4    11   15    0.4    84   59    9.7    6.9    0    157.1     1.0
27APR90 10:43:47     120    BWDLH58  9000/360  HP-UX 6.5    4    10   14    0.4    90   61   11.1    7.9    0    138.7     1.6
27APR90 10:45:47     120    BWDLH58  9000/360  HP-UX 6.5    1    12   13    0.3    90   62   17.3    8.6    0    120.6     1.8
27APR90 10:47:47     120    BWDLH58  9000/360  HP-UX 6.5    0    13   13    0.3    88   60   18.3    9.1    0    124.8     1.7
27APR90 10:49:47     120    BWDLH58  9000/360  HP-UX 6.5    0     6    6    0.2    86   60    8.4    5.1    0     88.3     1.0
27APR90 10:51:50     123    BWDLH58  9000/360  HP-UX 6.5    5     6   11    0.3    86   60    6.6    9.5    0     93.3     1.7
27APR90 10:53:48     118    BWDLH58  9000/360  HP-UX 6.5   12    20   32    0.9    93   62   17.5   13.2    0    357.2     2.3
27APR90 10:55:48     120    BWDLH58  9000/360  HP-UX 6.5    4    15   19    0.5    95   71   19.4   10.9    0    237.2     2.0
27APR90 10:57:49     121    BWDLH58  9000/360  HP-UX 6.5   32    39   71    1.9   100   86   48.1   15.6    0    537.3    14.3
27APR90 10:59:48     119    BWDLH58  9000/360  HP-UX 6.5   34    35   69    1.8    97   90   40.2   13.8    0    567.5    29.6
27APR90 11:01:48     120    BWDLH58  9000/360  HP-UX 6.5   29    25   54    1.4    97   91   17.4   11.5    0    518.5    15.3
27APR90 11:03:48     120    BWDLH58  9000/360  HP-UX 6.5   17    26   43    1.1    97   95   18.0    8.8    0    515.5     7.3
27APR90 11:05:48     120    BWDLH58  9000/360  HP-UX 6.5   21    32   53    1.4    77   63   23.3   13.1    0    565.0    18.7
```

A HEWLETT-PACKARD BASED AUTOMATION MEASURING SYSTEM IN THE FIELD OF SOLAR ENERGY CONVERSION

Dr. A. M. Marinoff, Bulgarian Academy of Sciences- Sofia

N. I. Georgieff, High School for Mechanical and Electrical Engeneering, Sofia

Mailing address: Bulgaria; 1618 Sofia; Peter Tonev Street;
Komplex Baxton; Block 26 W; Phone: 562310
Dr. Alexander M. Marinoff

Abstract:

Scientific research in the field of solar energy conversion requires experimental work, such as measuring of different physical values, on the one hand and theoretical work aimed at modelling, on the other hand. Theoretical and experimental work are interdependent in many aspects. This requires the usage of intellligent measuring systems characterized by high performance.

Most of the up to now concepts for realization of different physical measurements in the field of solar energy conversion did not suggest integrated solution.

The realization of a possible concept on a microcomputer measuring system for electrical, optical and temperature measurements in the field of photoelectrical and photothermal conversion of solar energy is discussed in the paper.

The measuring system is based on specialized microprocessor controlled instruments of the USA Hewlett-Packard company (digital voltmeter, LCR meter, programable power supply, scanner) and Perkin-Elmer company (UV/VIS- and IR-spectrophotometer) which are connected to a Hewlett-Packard microcomputer HP9000, series 80 through the interfaces IEEE-488 and RS-232-C. The microcomputer is completed with a standard microcomputer periphery of the Hewlett-Packard company (printer, plotter, 3 1/2"- and 5 1/4"- floppy disc drives and digitizer).

An application software, written mainly in HP BASIC for microcomputer HP9000, series 80 was developed for a wide range of electrical, optical and temperature measurements tasks in the field of solar energy conversion. It is possible to transfer easily the software, developed in the course of many years into HP BASIC ('Rocky Mountain' Basic) for more powerfull HP microcomputers (HP9000, series 200,300, 800), MICROSOFT QUICK BASIC or HT BASIC (High Tech. BASIC) for use on IBM PC/XT/AT.

The measuring system based on such concept is characterized by high performance, possibility for expansion

or new reconfiguration, easy handling and service, flexible software, compatibility to more powerfull measuring systems (for instance HP9000, series 200, 300, 800) ect.

It is also possible to develop the existing measuring system to an expert system for control of physical data in the field of solar energy conversion.

The measuring system can be used for control and physical measurements in laboratories and/or plants where solar energy convertors are designed or produced.

Keywords: Solar energy, solar system control, computer control, computer interfaces, software engineering, computer testing, programmed control, thermal variables control, optical variables control, electric variables control

1. INTRODUCTION

Scientific research in the field of photoelectrical and photothermal conversion of solar energy requires experimental work - for instance measuring of many and different electrical, optical and thermal values which characterize the functioning of the solar energy convertors (solar cells, solar collectors ect.). The experimental research is connected with the usage of different measuring units. This is due to the fact that the studied objects (for instance cristalline and amorphous thin films, insulating films, p-n- and Schottky-junctions, MOS-structures, solar water and air collectors) show a wide variety of physical behaviour. Thus the variety of measuring tasks makes difficult the control of the measuring units and the data acquisition. On the other hand, a precise modelling of various physical processes in the solar energy convertors is necessary for the correct interpretation of the physical processes which are still in a research phase.

The two ways for studying of physical phenomena - the experimental and the theoretical - are closely connected because very often a precise modelling is possible only in case that it is based on precise measuring data. For instance, an error of the order of 0.1 deg by the temperature measurement can lead to an error of 17 % concerning the estimation of the heat transfer.

From this point of view it is clear that the realization of a measuring system in the field of solar energy conversion is connected with a wide range of problems.

The most of the up to now used techniques for realization of different measurements in the field of photoelectrical and photothermal conversion of solar energy are connected with the usage of "stand-alone" measuring units in every specific case and do not offer the possibility to create an integrated solution both for experimental and theoretical problems.

In this work a realization of a possible concept on a microcomputer intelligent measuring system for electrical, optical and temperature measurements in the field of solar energy conversion is presented. The microcomputer system can be used also for theoretical modelling of physical processes. The measuring system is based on specialized measuring units and modules of the USA Hewlett-Packard and Perkin-Elmer companies which are connected through the interface IEEE-488-1 with a Hewlett-Packard microcomputer HP9000, series 80 to an integrated open system. The modern microcmputer technique allows for effective usage of the measuring units and the data acquisition. In practice this can be achieved in different ways. There are many possibilities to attain intelligent measuring system (Goetze, 1987).

A microcomputer (MC) is used as a central control and data acquisition unit. The microcomputer carries out the control, the measurement and the acquisition of the measured data. Besides, the different mesuring units are equipped with built-in microprocessors. In this case the microcomputer plays a coordinating role in the communication between the measuring units. The actual measuring tasks are performed by the intelligent measuring units. The software support of the measuring system includes system software for microcomputers HP9000, series 80 and application software for control, measuring, registering, data acquisition and modelling of a wide range of electrical, optical and temperature measurements developed by us.

The advantages of such concept for a microcomputer measuring system are as follows:

1. The usage of professional microcomputers allows for the automation of the measuring tasks, the data acquisition and the modelling of the physical processes.

2. The usage of the interface IEEE-488-1 allows for creating an integrated, open, high performance measuring system which can be easily widened or organized in a new way. Such measuring system is characterized by easy manipulation and service support, easy development of application software ect.

3. The software developed in the course of many years can be easily translated to more powerfull measuring and data acquisition systems (VME/VXI-bus- or IBM PC/XT/AT- measuring systems).

4. It is possible to develop the existing functioning automatic measuring system further to an expert system for control and measurements (Marinoff, 1989).

5. The measuring system can be used in research laboratories and industrial enterprises where solar energy convertors are developed or produced. But it can be utilized in many other fields of science and technique where different measuring tasks are carried out.

II. HARDWARE CONFIGURATION OF THE MEASURING SYSTEM

Fig. 1 represents a principle block scheme of an intelligent measuring system. It contains a MC complete with a standard microcomputer periphery (printer, plotter, floppy disc drive and digiizer) and measuring units (Marinoff, 1987a).

The 8-bit MC has an operating memory of 16k/32k or 128k (which can be expanded to 640k). The operating system (OS) takes 32k and it is realized on ROM's. The range of the OS can be widened with the help of 6 different ROM's. The MC is equipped with the well known measuring interfaces (IEEE-488, RS-232-C, BCD, GPIO ect.). This facilitates substantionally the control of the measuring units.

The measuring units include a digital voltmeter, programmable power supply (10A, 60V), multiplexer, LCR-meter and spectrophotometers for optical measurements in the range between 200 nm and 50 microns. The spectrophotometers are manifactured by the USA Perkin-Elmer company and the rest of the units- by the USA Hewlett-Packard company. All measuring units except the spectrophotometers have an IEEE-488 interface. The spectrophotometers have a RS-232-C interface and they can be controlled by the same MC.

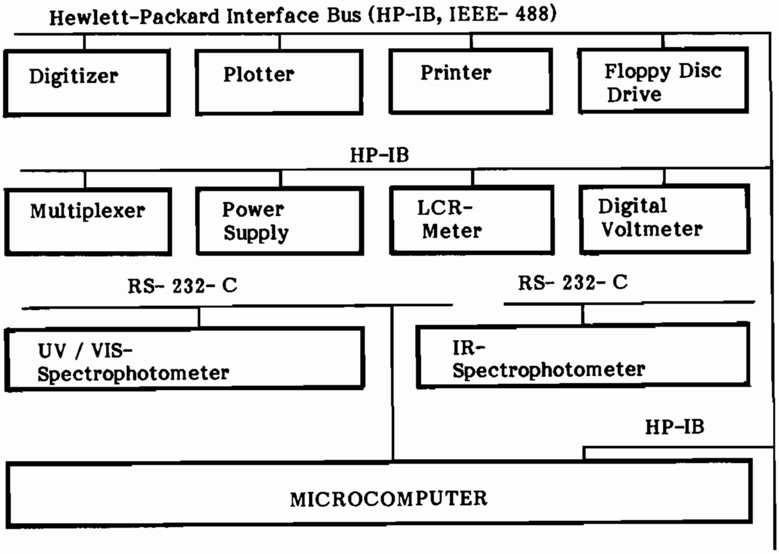


Fig. 1 : Principal hardware scheme of the measuring system

III. SOFTWARE CONCEPT

The above described hardware configuration allows for the usage of a measuring system in a wide range of applications. This is provided by both the hardware of every measuring unit and the system and application software. Appropriate hardware and software are selected according to the specific application of the system.

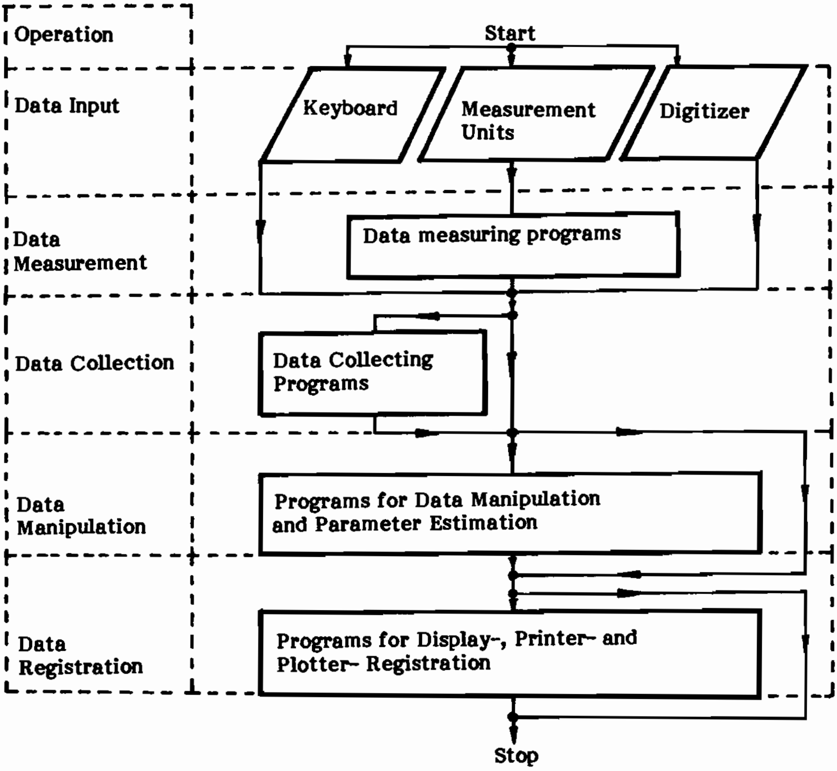


Fig. 2: Principal scheme for the program system design

Fig. 2 represents a common block scheme of the application software, developed by us. It is written mainly in HP BASIC and ASSEMBLER for the MC HP9000, series 80 from the USA Hewlett-Packard company (Marinoff, 1987b). Depending on the specific application of the measuring system the data collected from the studied object (for

instance solar cell, thin film or solar collector) is received by the computer in three different ways:
- directly through keyboard of the MC if the data is obtained from other units and it is necessary to be stored on magnetic media and registered in appropriate form.
- through the digitizer- for instance by estimation of optical spectres if they are obtained by spectrophotometers without digital output.
- the MC receives the data directly from the measuring units through the interface. This can be attained with the help of specialized program modules, developed for every specific case of application.

The software development is governed by several basic principles:

1. A complete application of the characteristics of the OS and the firmware for every data tasks (levels) (data input, measurement, collecting, manipulating and registering).

2. The programming is carried out successively for every level and is based on module principles. Different programming modules are developed for every level. These modules allow for the execution of diferent electrical, optical and temperature measurements in the field of solar energy conversion.

3. Some attempts were made in order to achieve standard modules for the different tasks.

4. There were other attempts made in order to attain very flexibile module organization for a specific data level (for instance for data registration). This could minimize programming costs for other measuring tasks. For instance, the programming module "data input from digitizer" used in optical measurements can also be used as data input module in electrical measurement tasks. The programming modules for data input from different measuring units have in principle the same structure. There are only unsignificant differences, which characterize the hardware of the specific measuring units used in the measurement. The programming modules for data registration are characterized by high flexibility and they can be used for different purposes with little variations only.

5. A minimum of hardware configuration is to be used for any specific measuring task, characterized by certain criteria. For instance, the measuring of V-A curves for semiconductor diodes and solar cells can be carried out in different ways in accordance with the requirements of the measurement. Hardware variations are possible in case of temperature measurements as well (Marinoff, 1988a). This can result in alteration of the software concept.

The described software concept was applied for the MC HP9000, series 80 but it can be translated to more powerfull measuring systems, such as VME/VXI or IBM PC/XT/AT based systems, because the latter have turned recently into industry standard. For instance, concerning more powerfull systems the MC HP9000, series 300 allows for widening the system to the VME-bus systems (Marinoff, 1988b). In this case it is possible to use IEEE-488- and VME- bus together and to utilize VME driver software for the operating systems BASIC, PASCAL, HP-UX (Hewlett-Packard variant of the operating system UNIX). It is possible to connect a VME-expander with 5 slots to the MC HP9000, series 300. Four slots are reserved for different VME-modules and the fifth slot is reserved for the VME-bus.

With the help of a MS-DOS coprocessor 80286 and appropriate driver programs it is possible to achieve program compatibility between IBM PC/XT/AT and HP9000, series 300 MCs. This leads to integration between the operating systems UNIX and MS-DOS and the usage of hundreds MS-DOS software products is possible.

There is another way to achieve flexibility of the above presented system: with the help of only one program driver and without any hardware modules it is possible to translate the developed application software and to transfer any data files from HP9000, series 80 to IBM PC/XT/AT working in the environment of MICROSOFT QUICK BASIC 4.0. Programs written in series 80 BASIC on MC HP85A, HP85B, HP86A, HP86B, HP87, HP9915 will be automatically translated to MICROSOFT QUICK BASIC 4.0 and run on faster MS-DOS MCs. MICROSOFT QUICK BASIC 4.0 is a fast powerfull language which can handle the complex and various programs created on HP9000, series 80. 100 % of the code of general computation programs will be translated. More than 90 % of the graphyc programs will be automatically moved to MICROSOFT QUICK BASIC 4.0. Hewlett-Packard Input/Output (HP-IO) commands will be translated as closely as possible to the library supporting the 82900A interface. The RS-232-C HP-IO commands will be translated to work with standard RS-232-C ports. The translator will automatically replace intrinsic functions. The advanced string handling included in HP9000, series 80 advanced programming ROM will be replaced by appropriate functions on the MS-DOS MCs. Matrix ROM functions handling complex manipulations of both matrixes and vectors will be supported. Translating functions of the plotter ROM will allow the user to continue using his graphic equipment in the MS-DOS environment. A MS-DOS file copy utility allows to copy data files and programs SAVED on HP9000, series 80 floppy disc drives.

Another way leading to further development of the existing measuring system is connected with application of the methods of the artificial intelligence, especially the technique of the expert systems and knowledge engineering (Marinoff, 1989).

IV. CONCLUSION

The discussed concept concerning the intelligent measuring system can be applied for solving a wide range of measuring and modelling problems because of the following advantages:

1. The MC HP9000, series 80 of the USA Hewlett-Packard company is a widely applicable controller in the field of computer aided measurement, test and process control. On the one hand, the MC itself represents a complete hardware and software system, on the other hand, it allows for the development of an open measuring system which can be easily widened. These MCs have been used for more than 8 years and their furthere development resulted in the creation of new series 200, 300, 500 and 800 of MCs. This provides compatibility of the different series from "lower to higher" levels of MC models so that the software used in the past can be easily transfered to the requirements of up to date MCs of the Hewlett-Packard company or to more powerfull systems such as VME/VXI- or IBM PC/XT/AT-MCs.

2. Every measuring unit of the system posesses a perfect hardware, system software and excelent reliability.

3. The measuring system is very flexible (besides, it is in fact an open system). That is why development of additional hardware is not necessary in many cases of application.

4. High level languages (BASIC, PASCAL, FORTRAN 77, FORTH) are used in the software development, and ASSEMBLER in case of necessity. The wide range of potentialities concerning the input and output operations with measuring units allows for a deep penetration into specific measurement problems.

The measuring system is used for a certain range of measurement tasks (concerning the scientific area of solar energy conversion), but it can be utilized in many other fields of science and technique.

The future application of the above described system will be connected with its widening to more powerfull measuring systems such as VXI, IBM PC/XT/AT or IBM PS/2 based systems.

The creation of an expert system on the basis of the currently operating automatic system for control seems to be even more attractive achievement (in terms of the future development).

V. REFERENCES

Goetze, B. and K.-H. Meusel (1987). Personalcomputer in der Messtechnik.Mikroprozessortechnik, Berlin, DDR, Heft 2, 54-56.

Marinoff, A. (1987a). Universal automatic system for electrical, optical and temperature measurements. Second National Conference with International Participation about Problems of Personal Computers "PERSCOMP'87". 21-24.04.1987, Sofia, Bulgaria.

Marinoff, A. (1987b). Software for intelligent measuring system for automation of scientific experiments. Scientific Symposium of the High School for Mechanical and Electrical Engeneering- Sofia "WMEI Lenin 87". 08-10.10.1987, Sofia, Bulgaria.

Marinoff, A. (1988a). In J. Zalewski and W. Ehrenberger (Ed.), Hardware and Software for Real Time Process Control, Elsevier Science Publishers B. V., Amsterdam, New York, Oxford, Tokyo, Chap. 4, pp. 145-150.

Marinoff, A. (1988b). On a concept for automation system on the basis of IEEE-488-1 bus with possible transfer to VME- bus systems. Fifth International School "Automation and Scientific Instrumentation'88", 11-21.10.1988, Varna, Bulgaria.

Marinoff, A. and E. Georgieff (1989). About a concept on expert system for control and interpretation of photoelectrical and photothermal solar energy convertors data. Second International Symposium of Socialist Countries "Theory and Application of Artificial Intelligence", 29.05. - 02.06.1989, Sosopol, Bulgaria.

Process Your Data on a Tight Budget - go UNIX

Julius Szelagiewicz
Turtle & Hughes, Inc.
1900 Lower RD.
Linden, N.J. 07036
(201) 574-3600

INTRODUCTION.

The company I work for, Turtle & Hughes is an electrical wholesaler and not a particularly large one at that. The company is run in a very traditional manner - the data processing expense is kept to the minimum, but a frugal management recognizes a dead computer system when it sees one. In 1987 it became quite clear that our hardware passed beyond the stage of being obsolete into the realm of the living dead. We were reaping the benefits of running a network - parts of it were always down. The response time was measured in minutes and we had to hire more people to enter data because of that. Also our hardware maintenance payments afforded us the weekly pleasure of a repairman's company. Finally it became obvious that the money paid to run the system could be put to better use. The data processing department (all two of us) was charged with finding a replacement. The budget was set at $200,000. The search for replacement, the installation of the new system and the subsequent developments follow.

The MISSION.

In the summer of 1987 our computer system consisted of network of old Datapoint computers supporting 35 terminals. There were 6 6000 series processors running terminal support (DATABUS programs running under DATASHARE running under Datapoint DOS) and network services (disk access only - no fancy stuff). The SNA communication for direct access to one of our supplier's IBM mainframe and the line printer each had it's own 6000 series processor. There were also 3 3600 series processors for development (slooow, but 24 line screens), and we had one IBM PC with an ARC network card. All of that lived off an ARC network running at staggering 2.5 MHz. The disk storage - five 60 MB CDC drives. The cartridges were removable, and a good thing too, because the tape was a very slow 1600 bpi. Each disk was divided into 5 logical volumes which meant that files could not exceed 12 MB in size. To do the backup (and we were unbelievably diligent about the backups!) we used mirror image copy provided directly by disk controllers. It took ten minutes per disk. Four disks can be backed up in less than 30 minutes when using 5 spindles. Lot of data was written out to tape never to be seen again. It was there, but where to put it on disk? Of course, the program interpreter would have nothing to do with the tape drive. The only pieces of equipment that never broke down where the Codex modems and multiplexers used to hook up the terminals in our Houston branch. The line was down often enough.

Because the equipment tended to break down very often, the juggling of the pieces was a daily occurrence. The processors were capable of running up to 24 terminals, that is, theoretically capable. Practically 10 was the limit. Remember now - those were 256K processors with virtual memory storage on a slow network. The multiplexers for terminals were also very easy to reconfigure - if you were handy with the soldering iron. I wasn't and I have scars to show for it. More often than not some non-critical part of the system wouldn't be up or the line printer would run slowly because the processor was needed elsewhere and the spare 3600 couldn't push data at 600 lpm. I had to write the unspooler in Datapoint assembler just to be able to drive 3 printers at speed. I have to admit that the $40,000 a year we paid for the hardware maintenance bought us the services of very cheerful and friendly repairmen. Frequent services.

The system was so slow that only necessary processing was done during the day. Any lengthy reports, heavy printing jobs or analyses were run on evenings or weekends. This, of course, necessitated having a full time operator.

The company got into this bind by happenstance. Twelve years ago, when the decision was made to computerize the operations a consulting outfit was called in. The management wanted a gradual approach, starting with accounts receivable. The consultant suggested buying a Datapoint computer and wrote the programs. The choice of Datapoint at the time was very good. They were the people with easy growth path and the screen oriented language. The setup worked just fine for a while. When I joined the company in 1982 there were seven terminals running accounts receivable, rudimentary order entry and inventory control. The software was being written by the consulting company. The problem was system availability. The programs tended to crash and trash data. It took the consultants hours to come and get the thing going again. Also there was the small matter of price. Hourly rates for software development are a little like renting a tuxedo for the deceased: you never stop paying. The system grew fast. Soon we had over twenty terminals. Once we had the network set up, adding more terminals was unbelievably easy. Just buy another processor with the terminal mux, hook it up and off you go - no need to stop the system. At the time it was extraordinary, because it worked and was cheap. Unfortunately the system slightly slows down with extra load. Programs were added and rewritten, the business changed and the software followed. We hired another programmer. The software actually got to a point where it was doing almost all that it was supposed to do all - the basic business functions save payroll and general ledger were included. The subsystems worked, even though they weren't deeply integrated. By that time our hardware was inadequate. The "almost all" meant that we couldn't do things that people take for granted on large systems - we had no on-line access to archival data older than a month, we couldn't use Trade Service data for lookup of non inventory items, we couldn't access proper item cross-reference from a terminal. Of course, comparing sales from last year with those of the year before, or projecting trends - little stuff known as management information was beyond our hardware capability.

Now we had to find a new solution. The obvious - buying better Datapoint hardware was out of the question. Datapoint was having big problems, the system software on their better computers was very bad, the hardware was expensive - the upgrade would cost us well over $150,000 and to add insult to injury, we would be forced to do a very tedious conversion of our programs. We were not too sure how long the hardware support would last, since it was spun off into a separate company, which also had big financial problems. We wanted to use the good stuff - fourth generation languages, databases, generators. In particular, we wanted to be able to run ORACLE. All this was out of the question on Datapoint. So now to ...

The SEARCH

We decided that whatever we do, we will not lock ourselves up in a proprietary computer system - being burned once was enough. (Of course the bosses wanted IBM, until they saw the prices) We were looking for portability and expandability. The company was growing at a slow but steady rate (5-6% a year) and the data processing requirements were growing much faster. There was much pent-up demand.

We started to look for a full replacement - software first. We've seen numerous packages, some of them had the option of customizing the programs, at a price of course. Most of the available packages were designed for a company much smaller than ours (by that time we were a $38,000,000 company). All of them would require either deep changes in our business practices or changes of the programs. We selected two most promising packages for deeper analysis. Alas, the price of software (with changes) combined with the price of hardware platform was well over $400,000 - out of our reach.

We have made three very consistent observations when looking for a software package.

First was that the programs were much more complex than necessary because they had to fit all kinds of businesses. Second was that for each function there was a distinct program (e.g., print and display of an order status would be done by be two separate programs). And the third was the fanciness of the screens. The screens were beautiful to the point of being too busy to remain readable.

Now, when you have severe hardware constraints you learn a way of very defensive programming - the minimalist approach works best, when well thought out. The first point - overcomplexity of the programs can be avoided when programming for one enterprise. But you don't want to go for a very tight fit - the business changes and the programs have to follow. It's better to plan ahead than to rewrite some very basic procedures - I learned the hard way.

The second problem - programs performing only one well defined function each, probably comes from a too literal understanding of modular design. There are good reasons to group like functions in one program. Some reasons are obvious - when you need to change order status layout only one display/print program needs work instead of two. Some reasons are less obvious, but extremely important. Consider the need to go though the menu to another program. The files have to be closed, screens displayed, new (sometimes same) files have to be opened. On our old system this kind of activity took forever. It is relatively time consuming on any system - so many system calls to open the files, get the buffers and so on. Well, the users didn't like waiting associated with going from program to program. We started at first tentatively and then vigorously to combine user related functions into programs. A good example of such combination is a program meant for salespeople that allows the user to check stock, enter order, check open or closed orders, check open purchase order, see info on supplier and on customer, find valid substitute material, and go directly to an IMS application on an IBM mainframe of our major supplier. Those functions take care of well over 90% of the salespeople needs.

Other stuff - mail, fax and such is provided by another program via a "hotkey" and is accessible to everybody.

The third point - busy screens. I love fancy screens, as long as their fanciness helps the users. But we are a simple business with employees of simple tastes. Pop-up windows scare us. And I have yet to figure how a pop-up window would really help a lady keying a batch of morning checks. Most data entry people look at the screen very rarely - the really look at the papers.

The screen that matches closely the format of the document helps a lot. The best and most beautiful screen with mouse and windows and banners and anything else you can think of will be a hindrance when the lowly document looks different. There is one more reason for the minimalist approach: serial data takes time to get to the terminal, lots of fancy data - lots of time. And the bandwidth is not cheap. Simple, clean screens display much faster.

Well, we didn't like any of the packages and we did like our application software and the direction in which it was going, so we decided to try to find a piece of software that would convert our code to C. That seemed like a good idea - we would later do development in C, or we'd use a program generator and the whole thing would certainly be portable. Unfortunately, the only conversion software we could find was not up to snuff. The code produced was unreliable, slow to execute (if at all executable) and completely unreadable. And now a piece of advice: Never, but never pay for a product you are just trying to evaluate if you don't know and trust the vendor. The money we paid wasn't reimbursed to us even though the contract stipulated full refund in no uncertain terms. We will get the money, or I should say, our lawyers will, but it came from my budget.

With translation to C fizzled, the remaining option was to find a DATABUS compiler. We found three that looked very promising - we talked to early users who were uniformly satisfied (we got their names from the vendors). One of the compilers would run only on a VAX, but the company was located geographically near - one hour drive. We started with that one. We still had a Microvax from our failed C conversion effort, so we tried the product (beforehand agreeing upon a reasonable fee for technical help - no prepayments). The compiler was very stable and for its hefty price ($25,000) delivered nicely. We were all set to go ahead and convert, our enthusiasm was dampened only by the fact that we would have to run on a Microvax cluster. DEC prices being what they were, the hardware capability shamelessly misrepresented by the salespeople, the cluster would be the only financially viable solution. Then a bombshell - there were no guarantees that the compiler would run on a VAXCluster. The network of Microvaxes was too slow for us. We went to test the other two compilers.

I tried the one from Subject, Wills & CO on our Microvax but there were problems. The other one, from Sunbelt Computer Systems, wouldn't run on VAX, so we borrowed an ATT 3B2 and tried it. It ran. But the thing was slower than our Datapoint. We tried it on an original IBM PC - was nearly 2 times faster, so back to the 3B2 to try in multiuser environment. It worked. The price ($6000) was OK, but changes required in the programs were much deeper that I was willing to contemplate. So back to the one that didn't run too well on VAX. At least it was cheap - $1,800. I went to Chicago to see one of the principals of the company - the guy that actually wrote the software (I did the same with other products). The man was unbelievably helpful, suggested I try it on anything - an AT running Xenix was his choice for an inexpensive test platform. He proved to me that programs run unchanged on a PC under DOS, under Xenix, on Novell network, on any UNIX platform (other than IBM AIX). The problem with VAXes was just being solved. The nicest thing was that the changes required in my programs were minimal - mostly tricks devised to squeeze every last drop of performance from the hardware. The test went OK.

The platform suggested by Don Wills was an H-P, as he put it: "their computers are much better than they know." I actually called an installation that was running a 9000-840 with 80 terminals using this compiler. The people sounded friendly and gave me file access timing results too good to be true.

Now I know better - they didn't tune well enough and because of that were running slowly. We talked to a few more users and decided that the least expensive is most beautiful. We talked some more to DEC people, but they were not interested in our order - too small. We sold our Microvax and started seriously looking for a platform.

Early on we decided to go to UNIX for a variety of reasons. We wanted portability, because one experience of being forcibly bonded to a hardware supplier was one too many. We wanted inexpensive system software - UNIX being an open system available from many sources should be really affordable, we felt. We wanted the tools that come with UNIX. Those tools are nothing to sneeze at. The data would be kept in plain ASCII format, so all the tools would be usable. We wanted a cornucopia of available third party software packages - databases, fourth generation languages, financial processing, communications, all of it cheap, of course. We wanted an operating system that is user friendly, and after dealing with IBM's OS/MVS anything seemed friendly to me. We also wanted something easy to learn, and with the sheer number of "how-to" books it seemed a cinch.

The ATT computers we could afford were very slow, and the fast ones were too expensive. We needed to put 42 terminals immediately (it's already spring 1988) - that was the current number of terminals on Datapoint, and we felt that 50 should do nicely for a while. We expected company growth of lessA card availabl only available solution (to the tune of $40,000) was a network with a 300 series computer running communication. Frankly, we decided to go with H-P even though it was more expensive because we have talked to people that were doing on H-P basically what we were trying to do, and mostly, because of Don Wills' endorsement - we trusted him. So we ordered a nice H-P computer 9000-840 just six months before it was phased out of production.

The system we bought consisted of a 9000-840 processor with 16Mb memory, two 7937 (571MB) disk drives, 7980A tape drive, 2564B (600lpm) printer, 8 additional 6 channel muxes (for total of 53 available serial ports), ThinLan network card. Because of small memory size, we decided to put the disks on separate HP-IB cards to better overlap disk access. In addition there was a 900-310 based system to provide gateway SNA services. The 840 came with the absolute minimum of system software - 64 user H-P UX license, networking software and SNA 3270 emulation. The 310 had a 2 user license, networking and SNA gateway software. The whole setup to the tune of $207,000. With $1600 for Subject, Wills & CO DATABUS, not too far over the budget. What we did not get was the 32MB memory - additional $15,000 was too much.

CONVERSION

We felt it would be prudent to do some really in-depth testing before the switch. Since our data processing department was two people we split the job into roughly equal pieces. I took care of planning and setting up the hardware, the system software, the communication and the application software and data layout. Our programmer was given the task of converting and testing of all programs and procedures, and of moving and setting data.

During the conversion a miracle happened - the users were actually nice to us, they cooperated, they were helpful. The level of frustration with the old equipment reached a point where any change was welcome.

We decided to keep our Datapoint terminals to cut initial cost and replace them with new terminals only at a time of breakdown. That really simplified the wiring work - the terminal end stayed unchanged. The communication with the IBM host wasn't established.

There was an insurmountable obstacle of communication protocol incompatibility. We decided to press ahead and worry about that later,after all how difficult can it be to buy an NRZI converter?

Most application programs required only minimal changes. In the few instances where much work was required, the cause was weird workarounds for the old hardware and system software constraints.

We decided to run a full-blown test on Accounts Payable. At the time it was the only module of the application that could be considered "stand-alone." The test involved mostly a parallel run for a week, including period closing. The results were excellent. We stopped running A/P on the old system and we looked at the calendar for the closest three day weekend. The Fourth of July weekend was the nearest. We had three full days and the Friday evening to move all the data, redo the wiring in the computer room, test day and month closing procedures on both systems, fix all new big problems (there weren't any) and be ready for the users on Tuesday morning.

The users had mixed feelings. They thought that a new computer means new terminals and new programs as well. They were delighted with the speed, but kept asking "Why do I have an old terminal when you keep telling me it's a new system." The fact that there was no retraining involved didn't register with them, but it sure was a boon to us.

What did we get from the Phase One (6 week conversion)?

Speed, speed, speed.

Savings. Monthly cost of the new system lease and maintenance is higher than the old one was, but we were able to reassign two people to different jobs.

High system availability - 23 hours a day (with 1 hour in the middle of the night when the end of day procedure is run).

Users who stopped swearing at the system and instead started swearing by it.

Problems, that showed up later:

Terminal communication: Datapoint terminals didn't support Xon/Xoff protocol, which would be no big deal if it wasn't for the fact, that we used many terminal printers. It turned out to be entirely impossible to use hardware flow control on our H-P. (You wouldn't believe the kludges that created.)

SNA communication: buying an NRZI converter turned out to be impossible - there was no market for them, because normally protocol is software selectable. Anyway, the matter was moot for almost a year - the H-P SNA software would not run on our 310. All this time we kept the Datapoint network running just to have the communication. (We finally borrowed the converter from the techies supporting the host.) The set-up is now different - since 310 is no longer supported, and the original SNA card couldn't possibly work in NRZI protocol, we have replaced the networked solution with a card in the 840, returning the converter to its owner. It took two years to get the SNA communication working, and the software is still wobbly.

System administration headache:

It takes much more time than anticipated to perform routine tasks that usually fall under the heading of "system administration". The need to keep the system going fast, takes precedence over new development, the questions of the users have to answered in "real time", the terminals and PCs break down, the communication links have hiccups, the power fails, the users erase their files on PCs and have no backups. The list is endless. In my optimism I thought that two hours a day should be more than enough for all of it. I was way off base.

Changes in the two years from installation.

Our predicted growth of less then 10% a year turned out to be more than 40% a year. We believe that enhanced data processing capability is one of the factors that enabled this growth. Even though the company added three new branches (that translates into addition of 40 new terminals), even though all batch processing was moved on-line, even though there are no restrictions on what jobs the users may run at any time, the response time is still satisfactory - on all data entry and most searches it is below one second. We don't even have a computer operator any more. The end users were trained how to print their special jobs (checks, invoices, etc.) and the backup takes about 20 minutes. We have managed to automate most of the daily functions that required human intervention. The end of day procedure is no longer interactive, because the system is reliable enough and fast enough to rerun it on those rare occassions when an error occurs. The fact, that the system can be easily monitored from home helps us enormously. We think that for the time being our biggest achievement was giving the users much better access to data.

Because of sufficient storage space we can keep data for the last three years. This allows us to use less paper and give users faster and more convenient access to information. The level of customer service improved dramatically due to computer speed and improved access to data. The order entry became truly on-line, the shipping is faster because there is no waiting between the time orders are received and entered (it used to take up to 3 days), most of the information about the status of the customer orders is readily available from the terminals.

We have moved aggressively toward specialized processing for our customers and vendors. Whatever the customers want - ability to order using their catalog numbers, special barcoded packing slips, any kind of reports - we happily promise it and deliver, provided the increased sales volume justifies our cost. This turned out to be an excellent sales tool. I'm being forced to go on sales calls. Special reporting for vendors gets us better prices, so we do it. This is where ORACLE comes in really handy. ORACLE is also indispensable all kinds of managment reports and analyses based on last few years worth of data.

To accomplish this we had to expand our computer system. We bought ORACLE license, we added an expansion cabinet and 7 6-port muxes, we also added another 7937 disk drive and an 8MB memory board. We upgraded our H-P UX license to unlimited number of users (and I think that the $15,000 price tag on it qualifies as an extortion). We have substantially upgraded our communications. The company now has three more branches, fortunately all locations are concentrated in two regions. The headquarters and one branch are located in New Jersey, one branch is in Houston, TX, another in Texas City, TX - 30 miles apart, and there is a small branch (a "twig", if you will) in Louisiana. To communicate with the NJ branch we use a dial-up 19200 Baud Codex modems and a pair of codex 6015 16 port multiplexers. The southern branches are connected via a 56K Baud digital link to Houston. The NJ end has a Codex 6742 32 port mux, the Houston end has same mux with 24 ports and an additional V22 bis card, which is used to connect Texas City branch via a pair of dial-up 19200 Baud Codex modems and one older Codex 6002 8 port multiplexer. The Louisiana branch dials with 2400 Baud modems into the 6002 mux in Texas City, because that's were we have free ports and the toll is lower than when calling NJ.

Where are we going?

We are trying to construct a virtually paperless system. In the last two years the company size almost doubled but paper use went down by a third. We managed to eliminate a lot of printing by allowing people to display the spooled print files on screen. It costs less to buy terminals capable of displaying readable 132 columns than to keep buying tons of paper.

We are doing our level best to speed up the customer service. The best way of doing it turned out to be providing well integrated access from the terminal to all historical data on customer - we want to eliminate completely the need to look for any paperwork when the customers call to inquire about an order or an invoice, and the paper trail sometimes can be very long. We are not yet where we want to be with it, but we are working on it. It is difficult to balance the storage requirements with the need to keep good access speed and not go wild with spending on hardware, but it seems that we will have to buy an optical disk.

In December last year we started to use EDI services on a PC. It's inexpensive and keeps our large customers happy. We hope to accomplish full integration of EDI with our system by year end. It turns out to be less simple and a lot more tedious at the same time than we suspected in our darkest moments, but it has to be done.

Some of our customers would like to be able to dial our computer, inquire about stock availability and place an order or ask for a quotation. We did have this capability on Datapoint, but the system was so slow, nobody wanted to use it consistently, so it died a natural death. Datapoint access was absolutely secure. Now I'm having nightmares about what might happen when we provide this functionality again. We will provide it, no matter how difficult it is to make it secure under H-P UX, because that will help us get business.

Our backlog is now just as big as it used to be, but the projects are different. We give much better response time to user requests, which spoiled them rotten. They request more and more and more often than not we grant their wishes. We squeeze many small jobs in between larger projects to resolve nuisances and productivity bottlenecks. We try to keep them happy by providing neat little things - ability to send a text fax from any terminal, new games ....

IMPORTANT FINDINGS

UNIX is IT! The choice of software and hardware was excellent (even if I say so myself). We feel secure in the knowledge that if we have to move to other hardware (if for instance we decide to decentralize the data processing) it will not be a very difficult task. We have actually tested the move to Xenix - works just fine. UNIX gives us a very rich set of tools to use on our data, it allows quick prototyping and helps us to easily take care of most ad hoc reporting requests. Unfortunately, all the tools are meant to be used with variable length records with fields separated by a special character. Of course, we use mostly fixed length records, and we miss out on some tools. (awk with length defined fields instead of character separated would really help). The system software is available in overabundance and it is much less expensive than for any proprietary operating system but it is not cheap. The tools for monitoring performance, that come with the system are woefully inadequate, and the good ones are really expensive. UNIX is not exactly easy to learn and use, but it beats OS/MVS hands down! Fourth generation languages and databases are fine and dandy, but if we were to rewrite our application in ORACLE we'd need about

$900,000 worth of hardware to run it at the speed we have now.

The hardware is very good. It is definitely better than we were led to believe by H-P salespeople. We were told that our 840 is good for 40 terminals, marginal for 50. At 80+ it is still running strong, and we will reach 100 soon.

The memory prices are astronomical, so we do it all in 24MB. I think that the hardware capabilities were consistently underestimated because H-P salespeople see only H-P software running on their computers. And that leads me to the Subject, Wills & CO DATABUS compiler...

The compiler/interpreter and associated tools exceeded our wildest dreams. On Datapoint I was all set to give up DATABUS as a viable language - too many restrictions, too slow to compile and run. But the DATABUS we are using now is unbelievably fast and very easy to use. It is a mature and sophisticated language allowing easy modular programming. And with well designed modules, program maintenance is much less of a nuisance than it used to be. The fact, that I can take our full set of programs and scripts and data and move it to any UNIX, XENIX, MS-DOS stand alone or networked platform without recompiling the programs is in itself unbelievable, but you should see the speed of indexed access, the floating partial generic key access.... I could wax poetic on that subject for quite a while, especially the index access that is better than any other I've seen, because its speed allows us to run so many terminals off our 840. This implementation of DATABUS also has a nifty source code level debugger and a bunch of other tools that come in quite handy - as opposed to UNIX utilities, they are based on fixed length fields and records, which is what we use most of the time.

Third generation languages are alive and kicking. ANSI standard means not only COBOL, C or ADA. DATABUS is in the process of becoming an ANSI standard language as well, and it just might be the ticked for budget minded. It mixes very well with C, it has embedded SQL access (SQL statements compile directly), one can even do device control from it. Of course it has full set of structured programing constructs, good set of math functions and powerful string handling. But it really shines in screen handling and in file access. I just love it to pieces.

You don't need GUI to check amount for accounts payable. Our users are quite comfortable with simple screens and system structure that closely follows the way the company does business. The fancy stuff is nice, but instead of increasing productivity, may lower it because of added layers of complexity. Also pushing graphics data takes a lot of bandwidth and graphics capable terminals or PCs, and that translates into real money. It really boils down to a question of when to say "good enough." We believe, that we found a frugal answer for a real business in the real world.

TITLE: Integrating OpenMail With Uniplex

AUTHOR: Garth Shepard
Uniplex
2000 Corporate Ridge Rd.
McLean, VA 22102
703-749-3668

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 1025

A MULTI-LEVEL COMPUTER COMMUNICATIONS SCADA NETWORK

By: Kevin B. Wong
East Bay Municipal Utility District
2127 Adeline Street
Oakland, CA 94607
(415) 835-3000 X776

ABSTRACT

A multi-level computer OPerations/NETwork (OP/NET) System provides Supervisory Control And Data Acquisition (SCADA) for a major Northern California water utility. The OP/NET System monitors and controls over 250 sites spread over 900 square miles.

Paired HP1000-A900's at five area control centers talk to 185 microprocessor based remotes and also pass data up to the paired A900s at Central Control. The Central Control also passes data to an IBM mainframe and DEC VAX minicomputer. Leased telephone lines and networks, direct cable, microwave, and multiple address system data acquisition radio provide the communications media.

This article describes the different computer and communications subsystems and how they all work together. System redundancy and high reliability of the system are also covered.

INTRODUCTION

East Bay Municipal Utility District (EBMUD) based in Oakland, CA serves an average of 220 MGD of water to 1.2 million people in Alameda and Contra Costa Counties covering 320 square miles. The OP/NET System monitors and controls:

- 165 reservoirs,
- 125 pumping plants,
- 24 rate control valves, and
- 6 filter plants

in the water distribution service area.
(See Figure 1A and 1B)

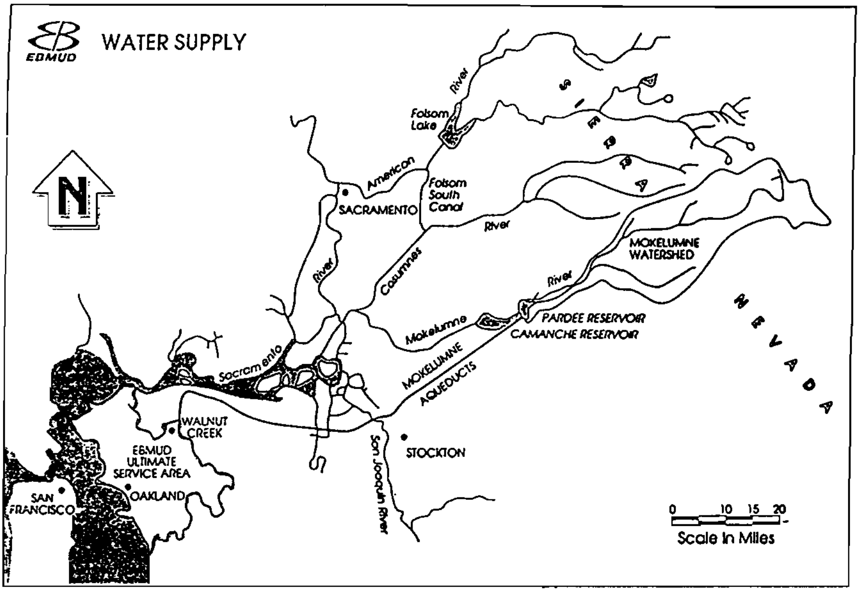


Figure 1A Water Supply

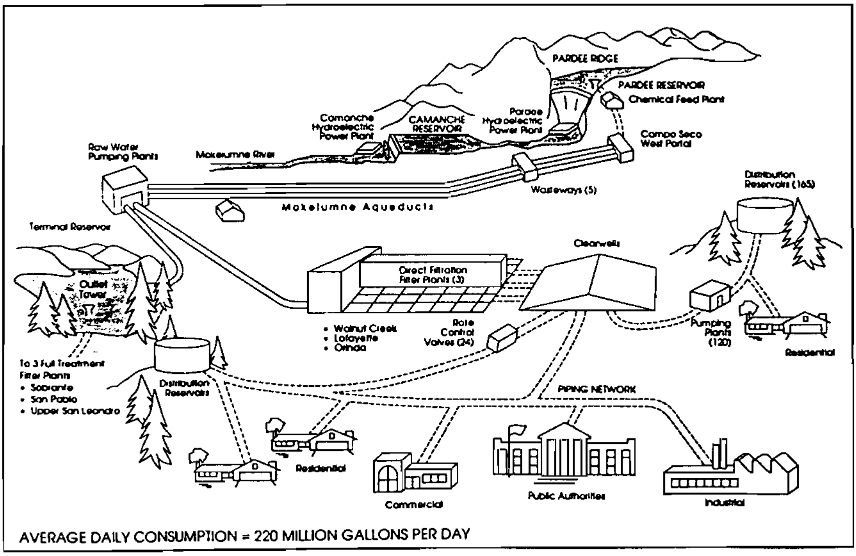


Figure 1B Water System Overview

The OP/NET system also monitors 5 weather stations in our 575 square mile watershed area in the Sierras and also controls:

- 2 Hydroelectric plants with 3 turbines each,
- 6 Wasteways, and
- 2 chemical feed plants along our 90 mile aqueduct system.

The system has been fully operational since January 1989.

OP/NET OVERVIEW

The OP/NET system has three layers (see Figure 2).

- The top of the pyramid is the Oakland Control Center (OCC). This is powered by paired Hewlett Packard, HP1000 series A900 mini-computers. A third A900 is used for system development.
- Paired A900s at five Area Control Centers (ACCs) provide the backbone for the middle layer of the pyramid and talk to the field remotes. One ACC is located at Pardee Center in the Sierra Foothills and 4 are at EBMUD's major filter plants in the Bay Area.
- Over 185 field remotes gather local data and provide control.

Communications subsystems provide the "ether" between the layers and consists of:

1. Leased multi-drop and point-to-point telephone company lines,
2. Multiple Address System Data Acquisition Radio,
3. Microwave Radio,
4. Direct cable, and
5. POTS (plain old telephone service).

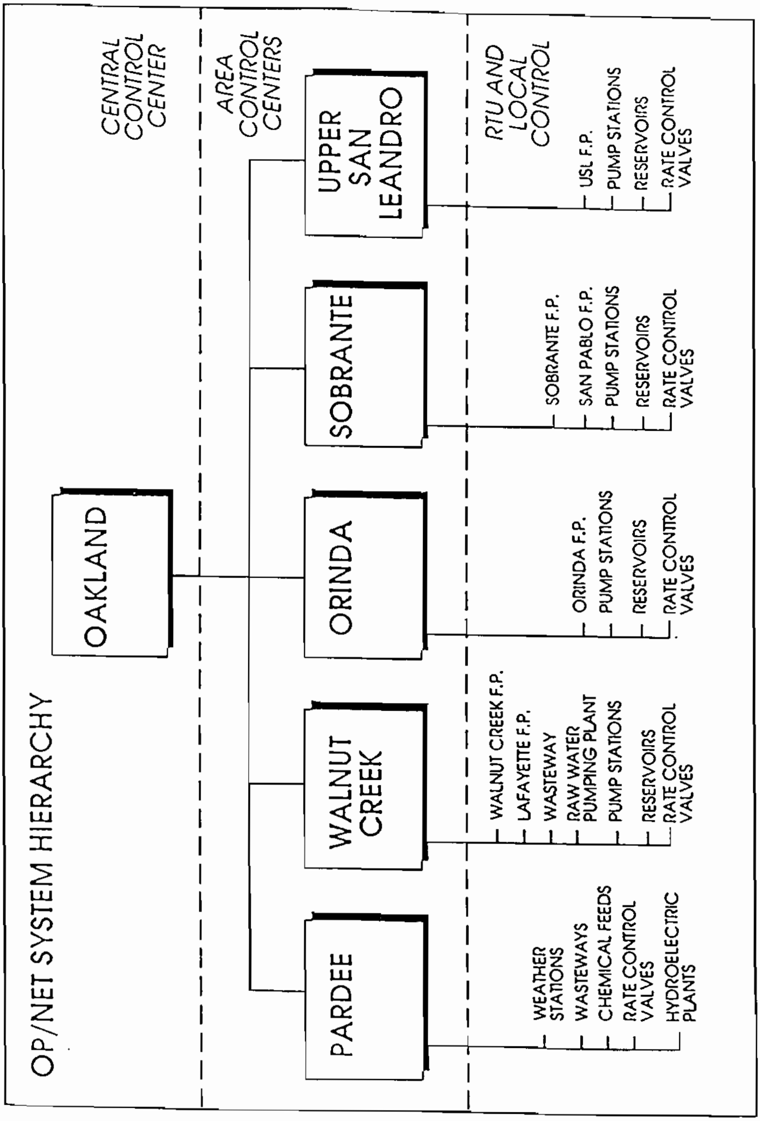


Figure 2 System Hierarchy

BOTTOM-UP DETAILS - FIELD REMOTES

The OP/NET System employs 7 different types of field remotes. The field remotes are not redundant but many have hard-wired back-up systems, such as electromechanical time clocks for starting pumps and pressure switches to shut them down. The field remotes are as follows:

- o 134 Hewlett Packard HP48000 Remote Terminal Units (RTUs)located at most sites such as pumping plants and rate control valves (see Figure 3).
- o 39 Advanced Logic Solutions: RTU-i500 (4 analog input & 8 discrete inputs) emulating a subset of HP48000 protocol mainly located at stand alone reservoirs. This is a single board RTU with an on board 202T modem.
- o 5 Allen Bradley series 5/15 Programmable Logic Controllers (PLC) for filter plant control at Orinda FP (expansion to add 8 more under implementation)
- o 2 Allen Bradley Series 2/30 Programmable Logic Controllers for Hydroelectric power generation plant control.
- o 5 Motorola INTRAC 2000 remotes with a master controller
- o 8 Steven Telemark units for monitoring river flow or reservoir levels - Dial-up units (under implementation)
- o 1 Larse Alarms system - for monitoring microwave system alarms (under implementation)

The field remotes are hardwired to field devices such as flow, level, and pressure transmitters, analyzers, status contacts, door and hatch entry switches, and motor START/STOP relays.

The OP/NET System currently monitors:

- 1600 analog inputs
- 2600 discrete inputs and controls
- 100 analog outputs (mainly used as set-points)
- 1200 discrete outputs

in addition the OP/NET System processes 1300 software points.

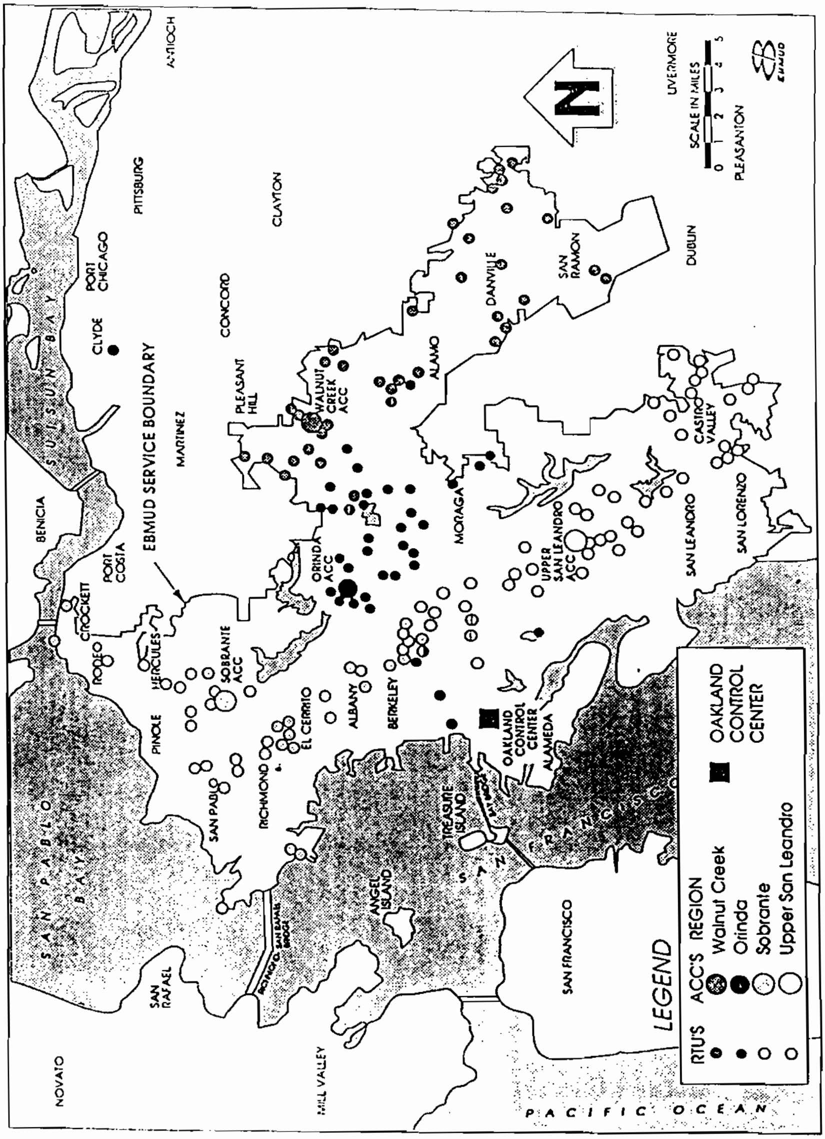

EBMUD SERVICE BOUNDARY
SUISUN BAY
SAN PABLO BAY
SAN FRANCISCO BAY
PACIFIC OCEAN
ANTIOCH
PITTSBURG
CLAYTON
CONCORD
PORT CHICAGO
CLYDE
PLEASANT HILL
MARTINEZ
BENICIA
PORT COSTA
CROCKETT
RODEO
HERCULES
PINOLE
SOBRANTE ACC
SAN PABLO
RICHMOND
EL CERRITO
ALBANY
BERKELEY
ORINDA ACC
WALNUT CREEK ACC
ALAMO
DANVILLE
SAN RAMON
DUBLIN
PLEASANTON
LIVERMORE
MORAGA
UPPER SAN LEANDRO ACC
OAKLAND CONTROL CENTER
ALAMEDA
SAN LEANDRO
SAN LORENZO
CASTRO VALLEY
TREASURE ISLAND
BAY BRIDGE
ANGEL ISLAND
RICHMOND SAN RAFAEL BRIDGE
SAN RAFAEL
NOVATO
MILL VALLEY
SAN FRANCISCO
N
SCALE IN MILES
0 1 2 3 4 5
EBMUD
LEGEND
RTU'S
ACC'S
REGION
Walnut Creek
Orinda
Sobrante
Upper San Leandro
OAKLAND CONTROL CENTER

Figure 3

COMMUNICATIONS - SUBSYSTEM - FIELD REMOTES TO ACCS

The HP and ALS RTUs communicate to the ACC via Bell standard 202T modems at 1200 baud. Most of the RTUs talk through leased 3002 4 wire full duplex telco multi-drop network. The phone company has an 829 loopback unit on each drop. Typically seven or eight RTUs reside on each network. Each RTU is typically polled for exception reports every 6 seconds. The ACC transmittals are heard by all the RTUs on the multi-drop networks. Only the "addressed" RTU responds to the host poll and this response is only heard by the host.

Multiple Address System Data Acquisition Radio (DAR) operate in the 928/952 megaHertz paired transmit/receive frequency band. The path must be line of sight and distances from the master range from 0.5 to 50 miles. Currently, up to nine remotes beam through the air waves to one master radio. OP/NET now has a total of 45 remotes talking to eight different master radios (see Figure 4).

The master radios are typically located on hilltops or ridges. The microwave system provides a relay between five of the master radios and the Area Control Centers. Two of the masters have leased telco line relays since Sobrante Filter Plant is not yet added to the microwave system. One is on site at Walnut Creek ACC, and is directly wired to the computer. The microwave system also provides a point-to-point links between the Allen Bradley 2/30 PLC at Camanche Power Plant and the Pardee ACC.

The five Allen Bradley (AB) 5/15 PLCs control 20 gravity/sand filters at Orinda Filter Plant. The PLCs are networked on AB's Data Highway Plus. An AB KF2 communication interface module hangs off the Data Highway and provides a serial port link to the Orinda ACC set at 9600 band. Additional 5/15 PLCS will be added at our Sobrante Filter Plant, Upper San Leandro Filter Plant, and Lafayette Filter Plant.

The Motorola Intrac 2000 AKA Hydronet, provides hydrological data from our 575 square mile watershed. The five remotes and three repeaters communicate with the base station at 169.425 to 171.850 megaHertz. The base station in connected via serial port to the Pardee ACC at Camp Pardee (see Figure 5).

The Stevens Telemark units will monitor river flows and reservoir levels. They will talk via Bell standard 212 modems at 1200 band when called on POTS (Plain Old Telephone Service).

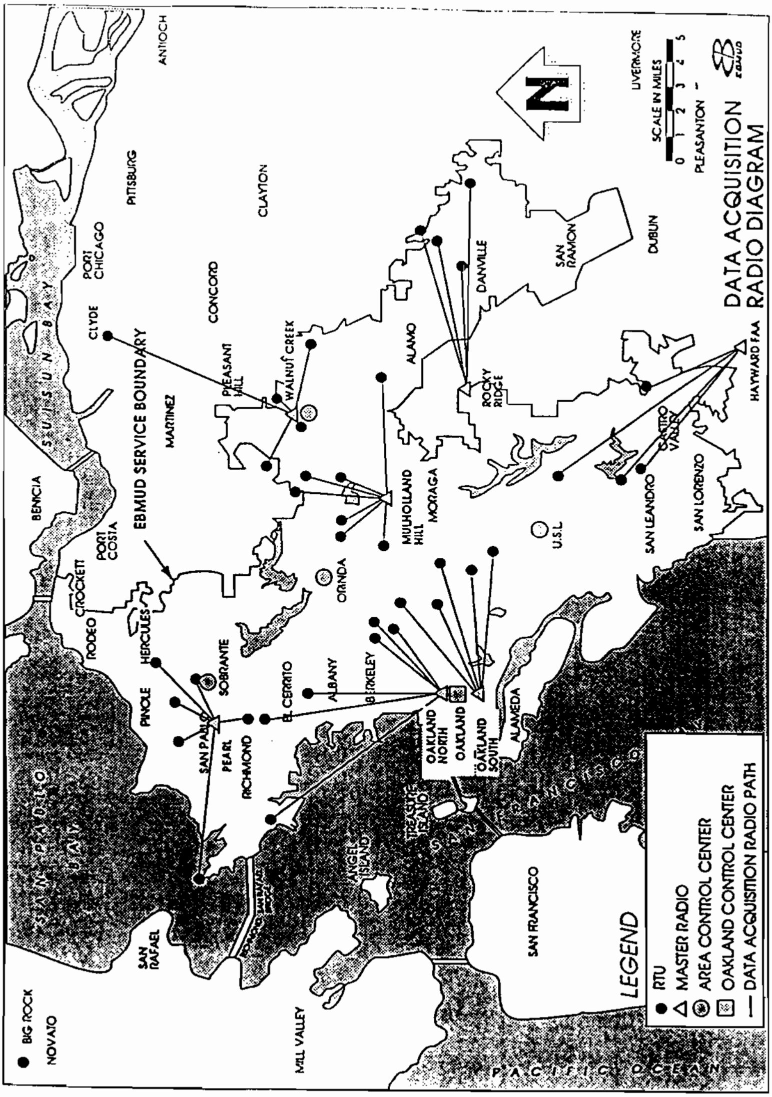

DATA ACQUISITION
RADIO DIAGRAM
EBMUD
N
SCALE IN MILES
0 1 2 3 4 5
LEGEND
● RTU
△ MASTER RADIO
◉ AREA CONTROL CENTER
⊠ OAKLAND CONTROL CENTER
— DATA ACQUISITION RADIO PATH
EBMUD SERVICE BOUNDARY
BIG ROCK
NOVATO
MILL VALLEY
SAN RAFAEL
SAN PABLO BAY
SUISUN BAY
BENICIA
PORT COSTA
CROCKETT
RODEO
HERCULES
PINOLE
SOBRANTE
SAN PABLO
PEARL
RICHMOND
EL CERRITO
ALBANY
BERKELEY
OAKLAND NORTH
OAKLAND
OAKLAND SOUTH
ALAMEDA
TREASURE ISLAND
ANGEL ISLAND
SAN FRANCISCO
SAN FRANCISCO BAY
PACIFIC OCEAN
MARTINEZ
CLYDE
PORT CHICAGO
PITTSBURG
ANTIOCH
CONCORD
CLAYTON
PLEASANT HILL
WALNUT CREEK
ORINDA
MULHOLLAND HILL
MORAGA
ALAMO
DANVILLE
ROCKY RIDGE
SAN RAMON
DUBLIN
PLEASANTON
LIVERMORE
U.S.L.
SAN LEANDRO
CASTRO VALLEY
SAN LORENZO
HAYWARD FAA

Figure 4

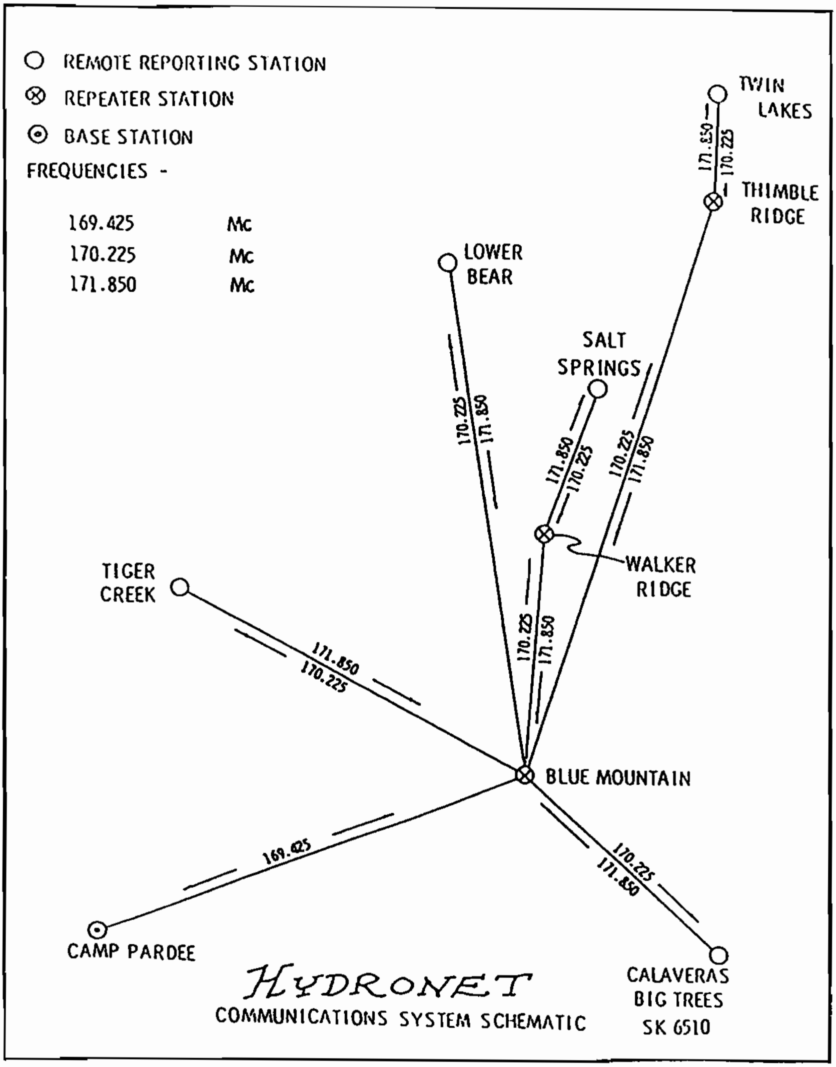

○ REMOTE REPORTING STATION
⊗ REPEATER STATION
⊙ BASE STATION
FREQUENCIES -
169.425 Mc
170.225 Mc
171.850 Mc
TWIN LAKES
171.850
170.225
THIMBLE RIDGE
LOWER BEAR
170.225
171.850
SALT SPRINGS
171.850
170.225
170.225
171.850
WALKER RIDGE
TIGER CREEK
171.850
170.225
170.225
171.850
BLUE MOUNTAIN
169.425
170.225
171.850
CAMP PARDEE
CALAVERAS BIG TREES
SK 6510
HYDRONET
COMMUNICATIONS SYSTEM SCHEMATIC

Figure 5

DETAILS OF AREA CONTROL CENTERS

The five ACCs all have very similar layouts. A generic description is the primary and back-up A900s each have:

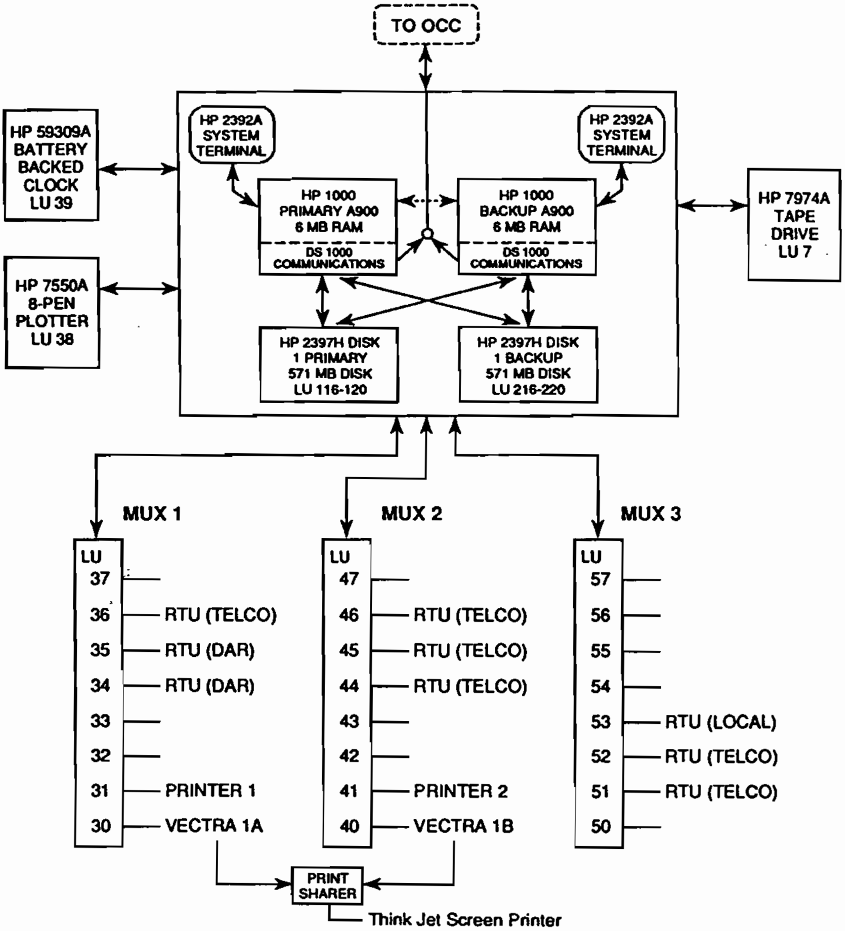
- 6 megabytes of RAM
- 3 multiplexer (MUX) boards w/8 serial ports each
- 1 DS/1000-IV network board
- 1 DI/DO card for watch-dog time-out switchover
- 1 Parallel Interface card for CPU-to-CPU communications
- 1 Async card for the system terminal
- 4 HPIB cards to talk to the disk drives, tape, plotter, and clock.

The 2 disk drives are a mirrored pair which are driven by HP's Datapair/1000 pseudo disk driver and can be accessed by either CPU. Software device drivers in the A900 handle communications and protocol translations with the field remotes.

The MUX channels meet together at an Excaliber auto switch box to which the I/O peripherals are attached. I/O peripherals hooked to the MUX are:

- UDS 202T modems (typically 9 for RTU communications).
- 2 HP Vectra AT compatibles which act as color workstations terminals for the operators.
- 2 printers for alarm, messages, and log reports.

HP's Distributed System DS/1000--IV hardware and software provide for communications with the OCC. The DS/1000 network is arranged in a star configuration with the OCC at the center hub.



TYPICAL AREA CONTROL CENTER HARDWARE OVERVIEW

(LU = LOGICAL UNIT)

Figure 6

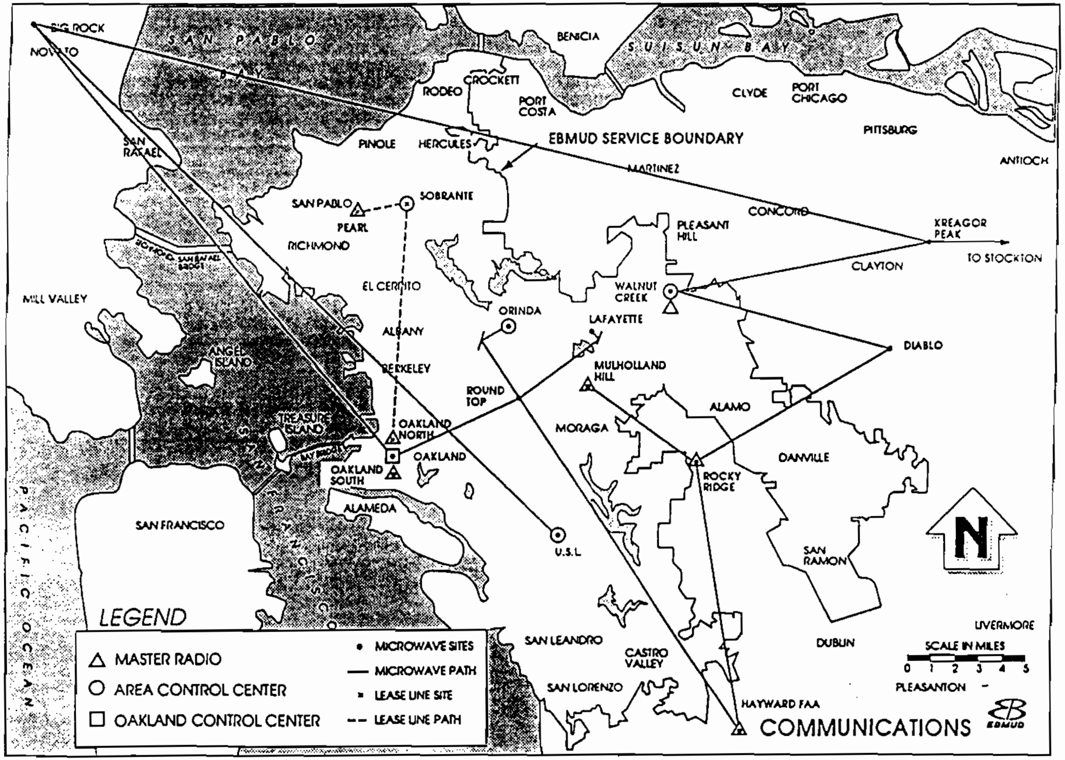


Figure 7 System Communications

The ACCs will automatically take over control for its area if communication is lost with the OCC (see Figure 6).
The back-up CPU sits in an idle state with the real time applications programs memory resident and ready to go. The back-up CPU monitors the primary CPU for trouble and automatically takes over in the event of a timeout. The switchover takes 1 minute.

MICROWAVE SYSTEM

The microwave system and leased back-up phone lines provide "4 wire" point-to-point communications between the ACCs and OCC. UDS 9600A V.29 compatible modems operate full duplex at 9600 baud (see Figure 7).

The microwave loop operates at 6.7 gigaHertz and currently has a 120 channel capacity. OP/NET uses 13 of these channels. Other channels are used for voice and data circuits. The microwave is also the backbone for our simulcast mobile radio system.

OAKLAND CONTROL CENTER DETAILS

The OCC is the core of our water distribution system. The OCC operators primarily control all the 125 pumping plants (over 320 pumps), 165 reservoirs, and 24 rate control valves from this site. Two operators are on duty during the day shift and one during the other two shifts.

Three A900s reside at the OCC. The odd A900 is similar to an ACC without the redundancy. It only has 1 mux card versus 3 and no DS1000 card. This A900 is totally isolated from the rest of the system and is designated as our training and development system. We use it for building graphic displays, building logs and software testing. The system also acts as a source of emergency spare parts.

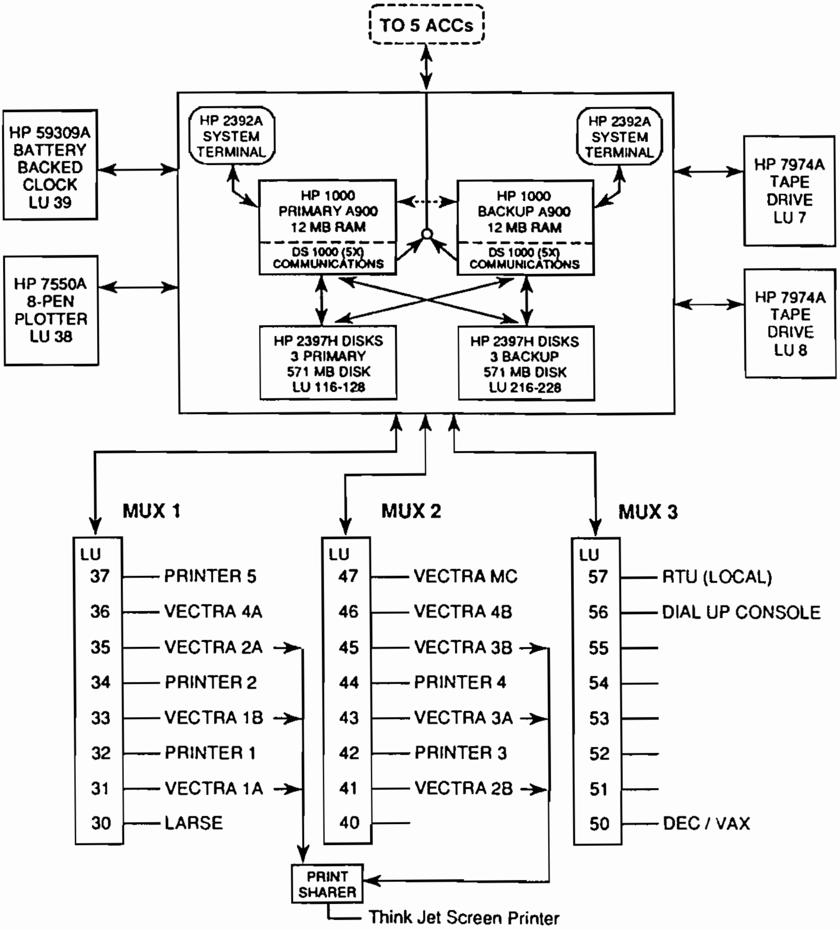
The other 2 computers are configured like the ACCs except for more (see Figure 8):

o 12 megabytes each of memory

o 6 571 MB disks (vs 2)

o 10 total operator stations (vs 2)

- 6 are in the main console
- 3 are individual manager consoles
- 1 is a dial-up



TYPICAL OAKLAND CONTROL CENTER HARDWARE OVERVIEW
(LU = LOGICAL UNIT) (DOES NOT INCLUDE TRAINING SYSTEM)

Figure 8 System Hardware Overview

- o 5 printers (vs 2)
- o 5 DS1000 cards each (vs 1)
- o 9 HPIB cards each (vs 4)

and less than an ACC: (only 1 modem for direct RTU communications)

Data is also transferred from the OCC to the IBM mainframe and the DEC VAX. Archived log files are transferred to a PC via Kermit protocol. The PC transfers the data to the mainframe which crunches it into an Oracle database.

RELIABILITY HISTORY

The 13 A900s have proven to be extremely reliable. We have only had (over the past 2 years):

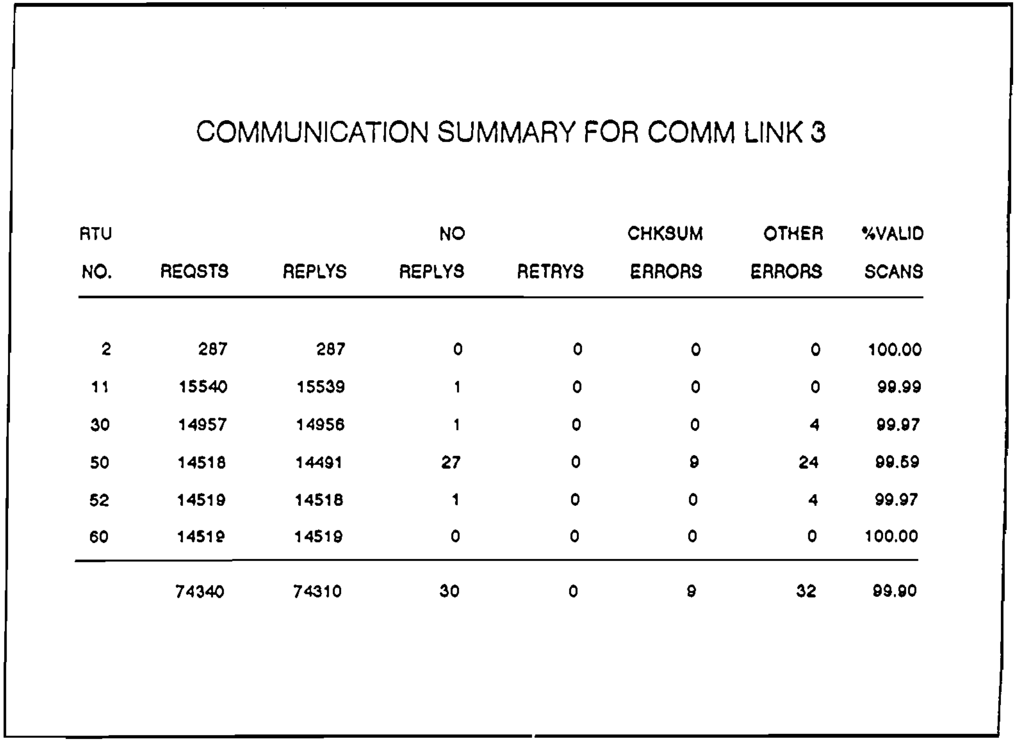
- o 1 memory controller board fail,
- o 1 read/write disk controller board fail, and
- o 2 system terminals smoke.

Hardware redundancy allowed continued operations in most cases short of a 7 minute reboot. The printers have had minor problems mainly with worn ribbon cables to the print head. The plotters have had minor problems with the paper feed rollers. The overall up time for the system hardware has been over 99.99% for the past 1.5 years.

Communications statistics are tracked and provided by a software module in the ACCs and OCCs. These statistics are a valuable tool in monitoring and troubleshooting communications subsystems performance. Many of our multidrop networks and point-to-point lines yield above 99.9% reliability. The statistics may be viewed on an hour-by-hour basis for previous and current day (see Figure 9).

CONCLUSION

Redundant CPUs and mirrored disk with robust and proven software have provided extremely reliable service and very high uptime. Software modules which monitor communication statistics are a vital tool in maintaining and troubleshooting communications subsystems.



COMMUNICATION SUMMARY FOR COMM LINK 3

RTU NO.	REQSTS	REPLYS	NO REPLYS	RETRYS	CHKSUM ERRORS	OTHER ERRORS	%VALID SCANS
2	287	287	0	0	0	0	100.00
11	15540	15539	1	0	0	0	99.99
30	14957	14956	1	0	0	4	99.97
50	14518	14491	27	0	9	24	99.59
52	14519	14518	1	0	0	4	99.97
60	14519	14519	0	0	0	0	100.00
	74340	74310	30	0	9	32	99.90

Figure 9 Communications Summary

ACKNOWLEDGEMENT

The high reliability and success of the OP/NET system is the result of great efforts over the last 3.5 years by many people at East Bay Municipal Utility District, Fisher Controls, and their local representative, Associated Process Controls, Hewlett Packard, ICF Kaiser Engineers, Westin Engineering, EMA Services, and several subcontracting firms.

REFERENCES

1. Wyman, Richard P., "A Three-Tiered Hierarchical SCADA System", ISA/89 International Conference and Exhibit Proceedings,, Instrument Society of America, Research Triangle Park, North Carolina, 1989, Pages 1267 to 1278.

2. Browne, David L., "Controlling a Large Water Distribution Process", INTEREX Proceedings of the 1989 INTEREX HP Users Conference RTE HP-UX, Workstations (San Francisco, CA, USA), INTEREX, Sunnyvale, CA, September 11-14, 1989, Pages 1001-1 to 1001-9.

3. Browne, David L., "Large OPerating NETwork Links 300 Facilities in Three Counties", AWWA Computer Specialty Conference Proceedings (Denver, CO, USA), American Water Works Association, Denver, April 2-4, 1989, Pages 521 to 532.

Automatic Logoff of Inactive HP 1000 Sessions

Donald A. Wright
Interactive Computer Technology
2069 Lake Elmo Avenue North
Lake Elmo, MN 55042 USA
612/770-3728

Abstract:

Inactive sessions left on a terminal can present problems in the areas of security and system resources. RTE's FMGR and CI operator-interface programs attempt to deal with the issue by providing their own inactivity logoff mechanisms. These often fail because they are not invoked by the system manager or just because the session is waiting for terminal response in some program other than CI or FMGR.

It is possible to develop a separate session activity monitoring program which provides an individual inactivity timeout for each RTE port, for either RTE-A or RTE-6/VM. This program can initiate a logoff of the complete session when the inactivity time is exceeded. A session is considered inactive if its accumulated CPU activity over the timeout period is below a specified minimum, and if there has been no terminal activity.

A major benefit is to prevent modems or terminals in sensitive locations from being left with an open session. The paper describes programs which are part of a commercial product, but there is sufficient technical detail for a good programmer to write his/her own version of the software.

The Problem:

Computer users DO leave sessions open. They may dial up and then disconnect without logging off (dirty disconnect); they may connect in through a data switch or LAN and disconnect or time out without logging off; they may use a terminal and simply walk away without logging off. They may even just shut their own terminal off and go home for the night. All of these open sessions can cause problems.

Security is the most obvious problem. An uninvited hacker (or worse) with access to a pre-existing session can produce by far the most serious consequences.

In addition, open sessions use system resources. At the very least, an idle session consumes memory or swap space for the session programs, some SAM or XSAM, and some ID segments. It can use scratch disk space, or have files open so that others cannot use them, or even have a data base locked.

The user interface programs CI and FMGR both have automatic timeouts, but these are only effective if CI or FMGR is the program that the user is currently running, and even then they may not provide the same timeout value that the system manager would prefer.

A Solution:

It is possible on both RTE-A and RTE-6/VM to develop a system-level monitor program which can automatically and absolutely delete idle sessions after a specified timeout (inactivity timer). That session timeout can be different for different ports and different types of ports. When a session times out, all programs in the session are deleted and the session is properly logged out.

The disadvantage of this approach is that a logoff of that sort is rather heavy-handed. It rudely aborts any programs in the session regardless of their state, and therefore it can destroy work done. But there are ways to minimize this disadvantage, explained later in the paper.

There is also a danger to the system from an automatic logoff. If RTE's logoff program (LOGON or LGOFF) displays any messages to the session terminal, it can be hung on I/O or even a down device if the terminal or dialup has been disconnected or turned off. Therefore it is important to perform the logoff silently, perhaps informing only the system console of the deed.

Determining Whether the Session is Inactive or Not:

Two techniques in combination are used to measure the activity of a session, and an automatic logoff should only be done if both measurements show the session to be inactive. They are:

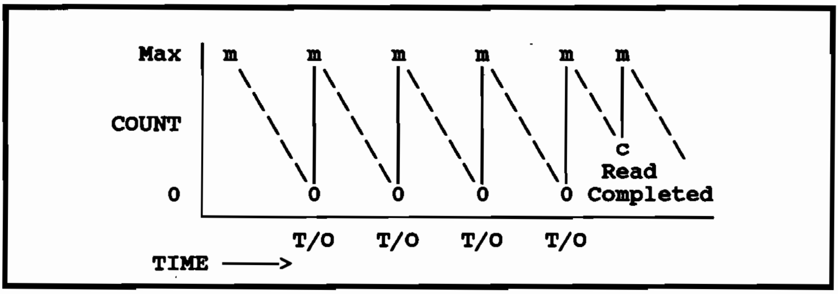
1. Terminal activity.
2. CPU Usage.

Measuring Terminal Activity:

RTE does not really provide any convenient hooks into the driver or DVT/EQT for reliably tracking or monitoring terminal read requests. However, it is possible to do an accurate job of measuring terminal activity by monitoring the terminal LU's timeout counter. That counter takes on a high value when each new I/O process is begun, and decrements with each clock tick down to zero (timeout). A graph of the timeout counter would look like Figure 1, below:

Figure 1

Running Timeout Clock



The completed read at the end is detected by observing that the timeout clock did not go to zero before returning to its maximum value. A program which needs to make this observation can put itself in the time list to examine the LU's running timeout clock value at least several times per LU timeout. It can look for a monotonic decrease in the clock value and ignore aberrations which occur around zero (the actual timeout). Any other departure from a monotonic decrease is a clear indication that an EXEC request has been completed normally and a new one started, which in turn is a very good indicator of terminal activity.

In RTE-6 systems it is very easy to sample the EQT's running timeout clock directly from a Fortran program. The EQT address is located via the LU table, and word 15 of the EQT is always the running clock.

The task is not nearly as easy in RTE-A. In order to save time when the clock ticks (100 times per second), RTE-A keeps all deviced in a timeout list. At each tick, RTE only needs to examine the first one in the list. To find the time left on a particular LU (running clock value), it is necessary to begin at the timeout list head and follow it to the DVT in question, adding the timeout increments from previous DVT's in the list. This is done with interrupts off, so that the system cannot change the list while it is being traced.

Unfortunately, the head of the timeout list (Q.TO) is in a system module other than VCTR, so that a program linked with access to Q.TO is not transportable to other system generations. While this is not desirable, it is a minor price to pay for the functionality received.

Here is a segment of code that can be used to chase the list and determine a particular DVT's clock value. Note: The address of DVT word 11 of the pending LU has previously been determined and stored in DV11.

Figure 2

Finding the Current DVT Clock Value

```
*   Shut down RTE-A:

        JSB $LIBR   Shut down the op sys (go privileged)
        NOP

*   Chase the linked T/O list to locate the LU entry,
*   keeping track of address of previous list entry:

        LDA Q.TOA   Use Q.TO address to init the chase
CHASE   SZA,RSS     If next-entry = 0,
        JMP OPSYON   we're done
        JSB .XLA1   Get next list entry from pending
        DEF @A       entry
        ELA,CLE,ERA Clear bit 15, just in case
        LDB A       B = DVT11 adr
        INB         B = DVT12 adr
        JSB .XLB1
        DEF @B      B = DVT T/O clock for pending DVT
        CMB,INB     Make it positive
        ADB CLOCK   Accumulate the clock total
        STB CLOCK
        CPA DV11    Is it the clear LU list adr?
        JMP OPSYON  Yes, we're done
        ISZ SAFETY  Emergency exit
        JMP CHASE   Keep looping

*   Scram, using $LIBX:

OPSYON  JSB $LIBX   Turn RTE back on
        DEF *+1
        DEF *+1
```

Another consideration when using the timeout clock to determine terminal activity is that the algorithm which looks for departures in the running clock's monotonic decrease will fail if the device does not have a timeout established, because the running clock will always have a zero value. There are two ways to deal with this issue, which will be most effective if used together:

1. The auto-logoff program can easily check each device in its list every time it runs, to be sure that a timeout is still established. If not, it is a simple matter to reestablish it by issuing the correct control request or even by forcing a value into the EQT/DVT either directly or by way of the system's TO command.

2. The detection algorithm can easily be arranged to fail in the direction of indicating NO terminal activity. This might tend to discourage people from setting their terminal timeouts to zero.

CPU Usage:

As with the timeout clock issue, determination of the session's accumulated CPU usage is much simpler in RTE-6 than in RTE-A. In RTE-6 that accumulated value is kept in the Session Control Block as a doubleword integer in words 10 and 11.

In RTE-A, the CPU usage is actually associated with each session program and is to be found in words 44 and 45 of the program's ID segment. Since there may be several programs in the session which have accumulated CPU time, it is necessary to search the ID segments for programs associated with the session and add up the contribution of each such program. Added to that total will be the value already accumulated in the User ID (session) table from programs which have previously terminated.

There is another issue in RTE-A: It is possible for the total of accumulated CPU usage among all session programs to actually decrease, because some programs (e.g. system programs like D.RTR) can accumulate time and then leave the session via XTACH or DTACH without posting their usage to the session's total. The algorithm which monitors for increases in CPU usage must account for this possibility.

One way to implement a CPU-activity measurement is to keep regular samples of a session's total CPU usage in a stack. Each time we make an insertion on top on the stack, we can look back in the stack by the amount of the session timeout, to see what the CPU usage was then. If the difference is less than the required minimum, then the session may be subject to deletion if the last observed terminal activity was also at least that long ago.

The reason for a minumum CPU usage limit, as opposed to a limit of zero, is that it is possible for an idle session to accumulate small amounts of CPU usage at each clock tick. The session program (CI, FMGR, EDIT, whatever) must do some small amount of work to examine each timed-out read and post another. Depending upon the program and the speed of the CPU, that may be enough work for the system to allocate a tick or two to the program at each LU timeout. These contributions to total session CPU usage are usually very small compared with any genuine usage, so the session usage limit can be set to a small enough value that there will be very little ambiguity about whether or not the session is actually idle.

Figure 3 shows a piece of Fortran code which follows the parent-child chain of programs from the primary to the end, adding CPU usage as it goes. IDA is the address of the system's first ID segment, IDN is the number of ID segments, and IDS is the number of words in a segment (all previously extracted from the VCTR module):

Figure 3

Accumulating RTE-A Session CPU Usage

```
C  Initialize the loop.  Be very sure the session's
C  primary program address is valid.  Note: we're doing
C  this with interrupts on, so the system can change
C  anything at any time:

      NAME  = ' '                   ! Preset
      USAGE = JXGET(SESADR+16)      ! From User ID table
      IDADR = IXGET(SESADR+19)      ! Primary program adr
      IDNUM = (IDADR-IDA)/IDS+1     ! Prim prog ID seg #
      GO = IDNUM.LE.IDN .AND. (IDNUM-1)*IDS+IDA.EQ.IDADR
      IF (GO) THEN
         NAME1 = IXGET(IDADR+12)    ! Zero if ID seg free
         GO    = NAME1.GT.2H
      ENDIF
      DO WHILE (GO)
         NAMEI(1) = NAME1           ! Copy in whole name
         NAMEI(2) = IXGET(IDADR+13)
         NAMEI(3) = IXGET(IDADR+14) ! In our session?:
         IF (IXGET(IDADR+39).EQ.SESADR)
     &    USAGE = USAGE + JXGET(IDADR+43)
         GO = IAND(IXGET(IDADR+15),77B).EQ.3 ! Waiting?
         IF (GO) THEN               ! Yep, get next prog
            IDOLD = IDADR           ! Save old ID seg adr
            IDADR = IXGET(IDADR)    ! Sanity check
            GO    = (IAND(IXGET(IDADR+14),377B)-1)*IDS+
     &      IDA .EQ. IDOLD
            IF (GO) THEN
               NAME1 = IXGET(IDADR+12) ! Next prog valid?
               GO    = NAME1 .GT. 2H
            ENDIF
         ENDIF
      ENDDO
```

DOWN Device:

It is also possible (and easy) to examine the pending LU to see if RTE has set it DOWN. If so, this can indicate (particularly with C-MUX LU's) that the ENQ/ACK handshake or XON/XOFF pacing has failed. Usually this is because the user has disconnected the terminal from the port or has turned it off. In either case, it may be reasonable on some systems to assume that sessions with a DOWN session LU no longer have a terminal attached, and to delete them as if they were timed out (idle).

When this is done, the port should be brought back UP with a MESSS request after all I/O clears, so that it will be available for use when a terminal reconnects or is turned back on.

Logoff Mechanism:

Logging off the session cleanly does involve a few tricks, some of which have to do with the need for a silent logoff which will assure that nothing gets hung trying to write to a dead port.

This time it is RTE-A which makes the job easiest, through a call to the RTE library subroutine CLGOF. This subroutine will log a session off cleanly, posting accumulated session totals as it should. If the session number is greater than 255, CLGOF assumes that it is a programmatic session and performs the logoff silently. Therefore the auto-logoff program can change word 12 of the session's ID Table entry (session number) to a high value just before calling CLGOF, and then the logoff will be silent.

A good deal more work is required on RTE-6. First the LGOFF program's class number is located in the $VCTR entry point $LGOF. Then the session LU is temporarily reassigned to EQT 0 (for silent logoff) and LGOFF's priority is temporarily changed from 90 to 29 so that the logoff will occur quickly. Then LGOFF is scheduled via a write to its class number.

LGOFF will do a very competent job of deleting all session programs and logging any accumulated times. The auto logoff program may wait for the session to disappear and then reassign the correct EQT to the session LU and the correct priority to LGOFF.

A caution: LU 1 (the system console) must be treated as an exception; its EQT should not be changed. If LU 1 is assigned to EQT 0, RTE-6/VM will stop.

An example of RTE-6 code is shown in Figure 4, next page.

Restoring Ports:

When a session has been deleted, it is reasonably likely that the associated port (LU) is not in a normal state. It may have been sitting in a screen program such as a LAS program, or it may be DOWN, or it may be disabled. Since it is a port which should be enabled for logon (or it would not have had a session on it), it probably needs to be reset and re-enabled.

What is done to reset the port may vary quite a bit from one system to another. On some systems it may be desirable to duplicate the function of the WELCOME file, including the full 33B and 34B requests, to reestablish the port as a standard HP protocol port. On others it may be better to use the SID program called INITMUX (or INITM), which can do about anything one might need. At the minimum, a 20B (reenable) request should be issued.

Figure 4

Scheduling LGOFF on RTE-6

```
         IF (OLDSES .NE. 1) THEN
            LGIDSEG = IDGET(5HLGOFF)      ! LGOFF's ID seg
            IF (LGIDSEG .GT. 0) THEN
               LGIDSEG = LGIDSEG + 6      ! Priority word
               PRILG   = IXGET(LGIDSEG)   ! LGOFF's priority
               IF (PRILG .EQ. 90) THEN    !  gets changed
                  CALL IXPUT (LGIDSEG, 29)!  to 29
               ELSE                       ! Unless not 90
                  LGIDSEG = 0             !  then set flag
               ENDIF
            ENDIF                         ! Make EQT = 0:
            CALL REASSIGNLU (OLDSES, 0, OLDEQT)
         ENDIF
         CALL EXEC (100012B, 5HLGOFF, *22)! Wake up LGOFF
   22    LGCLASS=IXGET(ADDRESSOF(LGDUMMY))! LGOFF's class #
         CALL EXEC (140024B, 0, 0, 1,     ! Write to it . .
        & IOR(020000B,OLDSES), SESADR, LGCLASS, *44)
   44    IF (OLDSES .NE. 1) THEN          ! Skip sys console
            CALL EXEC (12, 0, 1, 0, -10)  ! 100-mS pause
            TRY  = 1                      ! Session gone?:
            DO WHILE (TRY.LE.4 .AND. LUSES(OLDSES).GT.0)
               CALL EXEC (12,0,2,IVL, -3) ! Be swappable
               TRY = TRY + 1              ! Try for 12 sec
            ENDDO
            CALL EXEC (12, 0, 2, IVL, -3) ! LGOFF cleanup
            IF (OLDEQT .GE. 0)            ! Restore EQT
        &    CALL REASSIGNLU (OLDSES, OLDEQT, I)
            IF (LGIDSEG .GT. 0)           ! Restore LGOFF
        &    CALL IXPUT (LGIDSEG, PRILG)  !  priority
         ENDIF
```

Minimizing the Damage:

The most important way to minimize the potential of lost work caused by the arbitrary deletion of sessions is to implement the new monitor program as gently and gracefully as possible, giving plenty of warning to users and also generous timeouts, especially at first. This also reduces the system manager's risk of personal injury.

If you have been living with the possibility that dialup sessions could be left open forever, then the difference between a session timeout of 2 minutes versus 20 minutes does not have a large impact on the change you are already making in the system's vulnerability, but it could make a lot of difference to the user who is accustomed to having the time to go get a cup of coffee. When the come to appreciate the logic of an inactivity timer and have become used to it, then it may be reasonable to tighten the timeouts a bit more.

Training is important, even though it may not require more than several minutes per user. They should know not only what is in store for them if they leave a session idle but also what to do if they are logged off. As a minimum, they should know how to properly handle the recovery procedures available in programs such as EDIT and CCWORD.

It is possible for the auto logoff program to try to warn a user, by writing to the session LU a minute or two before shutting the session down. This can be dangerous if not done very correctly, because the program doing the warning can be hung on pending I/O or even a down or locked LU. Preferably, all of these mechanisms would be used together to reduce that danger:

1. Class I/O write.
2. Write-through-pending-read bit set (RTE-A).
3. No-wait bit set.

There may also be cleanups that the auto-logoff program can perform which will reduce the damage. This is highly dependent upon the programs running on the session when it is deleted. An example: The program could scan the /SCRATCH directory for EDIT scratch files created by the deleted session and rename or move them to another location for safekeeping. On a system with MAIL, the program could even write a message to the logged-off user telling him/her where the recovery file is and how to access it.

Summary:

Using the techniques described above, it is a relatively straightforward matter to develop a system-level monitor program which will watch for and delete inactive sessions.

Advances and Trends in Mass Storage Technology

Husni Sayed
Andrew Patterson
Deborah Cobb
IEM, Inc.
P.O. Box 8915
Fort Collins, CO 80525
(303) 223-6071

Introduction

Since the introduction of magnetic tape as a storage medium, mass storage technology has been advancing steadily in an attempt to keep up with the demands of users. The development of Winchester disk technology introduced the computer world to higher capacities, more reliable storage, and faster access to stored data. Following this revolution in data storage techniques, advances in mass storage technology slowed considerably: until the last decade. Recent breakthroughs in the technology, popularizing Removable Winchesters, Optical disk drives, and high capacity tape systems, are defining the shape of the future.

Looking Back

Magnetic tape systems and Winchester disk drives, developed not too long ago, are already beginning to fade into innovations of the past. While these technologies will certainly have a role in mass storage for many years to come, the push for faster, smaller, more reliable systems is already starting to shoulder these solutions out of the limelight.

Magnetic Tape

Magnetic tape, still in use in many places, has been used as a backup and archival mass storage medium for decades. Magnetic tape systems in use up until recently are essentially the same as when they were first introduced: the technology has advanced very slowly, especially in comparison to other mass storage techniques. Although tape storage capacities have increased dramatically with the introduction of thin-film media, access times (which are a function of tape speed) have not changed much over the years. Tape densities have increased, but slowly.

Before the advent of Winchester disk technology, tape drives were used (out of necessity) as primary mass storage devices. Though not highly efficient, the only alternatives at the time, such as paper tape, were even worse. In an attempt to produce acceptable random-access times, multiple tape drives were used to access a single logical file. Today, with the wealth of faster, more efficient primary mass storage alternatives, tape drives are used almost exclusively for backup, archival, and data interchange purposes.

In addition to being slow, there are other limitations that are inherent in tape media. Due to problems with media wear, a single tape typically cannot be rewritten more than 300 times. Because of problems with stretching, breaking, and print-through, the data retention time for magnetic tape is less than 3 years. Magnetic tape also has a limited storage capacity in comparison to newer mass storage technologies. A $10\frac{1}{2}$" 9-track tape reel can store only about 180 MBytes at maximum density.

9-Track Tape

9-track tape uses large, open reels to hold the $\frac{1}{2}$" tape. The term "9-track" refers to the manner in which the data is stored: in nine distinct tracks across the width of the tape. One track is used for each of 8 data bits, and the remaining track is used to store a parity bit.

Advances in 9-track tape technology have focused primarily on increasing the density of stored data. The first 9-track tapes had densities of 550 bytes per inch (bpi), while today's high-end 9-track tapes boast densities of 6250 bpi.

The single greatest advantage to 9-track tape is its pervasiveness. Virtually every computer installation has at least one 9-track drive somewhere. In addition, the data formats have been highly standardized: all 9-track tapes of equal density write and read data in an identical manner. These two traits have made 9-track tape the medium of choice for data interchange tasks. For this reason, 9-track tape drives will continue to be popular until another standard of data interchange emerges.

Cartridge Tapes

With cartridge tape, a small plastic case is used to enclose two reels of tape. The tape is automatically loaded when the cartridge is inserted into the drive. Cartridges have several advantages over 9-track tape, including their compact size, ease of handling, and resistance to the environment. There are two types of cartridge tapes commonly in use today: the 3284 cartridge and the $\frac{1}{4}$-inch cartridge.

The IBM 3284 (or, simply 3284) cartridge is named after the IBM computer for which it was developed. The small cartridge, approximately 4" by 4", uses $\frac{1}{2}$" wide tape. It stores data in a manner similar to 9-track tapes, except that multiple 9-bit tracks are stored across the width of the tape. Currently, 3284 cartridges use two 9-bit tracks, though it is expected that this track density will double several times in the future.

QIC, or Quarter Inch Cartridges, are roughly 6" by 4" and (as the name implies) use $\frac{1}{4}$" wide tape. Smaller sizes are available for personal computer systems. QIC tapes have capacities ranging from 40 to 135 MBytes, depending upon the length of the tape.

QIC tape drives employ serpentine serial recording. Data is written in 9-tracks down the entire length of the tape. At the end of the tape, the direction of tape motion is reversed and the data is then written parallel to the first set of tracks, but in the opposite direction. A major problem with QIC tape drives is their lack of standardization, as each manufacturer implements storage schemes in their own way.

Magnetic Disk Media

Magnetic disks can be divided into two general categories: Removable disk media, and Winchesters. Magnetic Winchester hard disks were originally developed in response to an outcry for *faster* access to stored information. These random access devices, much faster than the sequentially-accessed tapes, finally gave users access to data in under a second.

Early in their evolution, Winchester disks acted mainly as "middlemen" between tape systems and main memory: users continued to rely on magnetic tape for storing data, and transferring information between computers. As Winchesters became faster, smaller, and more reliable, they gained more widespread acceptance and use. Today, these mass storage devices are found virtually everywhere, offering capacities anywhere from 10 to 2000 MBytes. Though more compact storage methods have since been developed, no other solution has been able to beat the Winchester where speed is an important consideration. Manufacturers have recently developed "Remvovable" Winchester drives consisting of a main drive unit, with a hard disk unit that can be inserted into the main unit and removed as necessary. This gives the user the ability to remove the disk for storage in a secure location, or purchase additional media without the need to buy a whole new disk drive.

Removable disk media are relatively small, thin magnetic disks or disk cartridges that are inserted into a drive unit. This class of magnetic disks includes the ever popular "floppy" disk, and other technologies such as the Bernoulli disk. Removable disk media do not offer as high a capacity as Winchesters, and most have slower access times and data transfer rates.

Current Trends

Helical-scan tape and optical disk drives are transforming mass storage technology. Helical-scan tape is inexpensive, and has a large enough capacity to allow unattended backup of on-line systems. It would take almost 13 reels of high density (6250 bpi) 9-track tape, or 12 IBM 3284 cartridges, to store as much information as a single 8mm tape cartridge. Furthermore, the traditional tape system would require the presence of an operator to change every one of those reels or cartridges: a process that may take hours. This additional labor cost makes helical-scan tape even more attractive. Decreased storage space is another incentive that favors helical-scan tape. 8mm tapes can store 326 MBytes per cubic inch, compared to 3 MB/in^3 for 9-track tape, or 27 MB/in^3 for QIC tape.

Optical disks have many of the advantages of helical-scan tape, such as large capacities in small volume, and also offer faster access times. They are serving to fill a previously empty niche that existed between large capacity, low cost mass storage tape systems, and fast access, high cost Winchester disks. The removable nature of optical media makes it ideal for storing large software systems and data bases that can be swapped in and out as needed. Erasable opticals allow for instant access to on-line removable file systems. This makes them ideal for large, less frequently accessed data bases. Winchesters will still be in demand where fast access times are crucial however, such as in virtual memory systems.

Helical-Scan Tape

Helical-scan tape stores data using technology that was originally developed by the video recorder and digital audio tape industry. The name is derived from the method by which the tape travels over the head. Previous tape technologies used a fixed head, with tape passing (relatively slowly) over the head. Helical Scan, however, uses a head that is mounted on a rapidly spinning drum aligned diagonally to the track. As the tape passes over the drum, the head writes tracks of data in a diagonal pattern corresponding to the pitch of the head. This method produces track densities on the order of 1,000–2,000 tracks per inch.

Although helical-scan drives used in the computer industry are very similar to a VCR (Video Cassette Recorder) or a DAT (Digital Audio Tape) player, they require a much higher reliability. VCRs and DAT players typically have an error rate of 1 in 10^6. When an error occurs during a VCR recording, a small extraneous spot may appear on the screen. Similarly, in the case of DATs, a timeout may occur for less than a millisecond. These types of errors are virtually undetectable to human senses. An error rate of 1 in 10^6, however, is unacceptable for mass storage applications. Because of this, error checking and redundancy must be implemented to produce an error rate more in the neighborhood of 1 in 10^{13}.

8mm Tape

8mm helical-scan tape systems, derived from commercial Camcorder recording technology, are a very recent addition to mass storage applications. With a maximum storage capacity of 2.3 GBytes per tape, 8mm media has the single highest storage capacity-to-volume ratio of any mass storage device currently in use (326 MB/in^3). Access time, as with all tape systems, is relatively slow: in the tens of seconds. Burst transfer rates are on the order of 10 MBytes per minute.

The media used for 8mm tape in the mass storage industries is the same lightweight, plastic cartridge used in the entertainment field. Any high quality metal tape from a Camcorder can be used in an 8mm helical-scan tape drive. The 6" by 4" by $\frac{1}{2}$" cartridge fits easily into a shirt pocket. The volume of sales for the entertainment industry has drastically lowered the price of these cartridges to under $10.00 each. This factor, plus the huge storage capacity of these tapes, has driven the cost of storage to less than one penny per MByte.

Digital Audio Tape (DAT)

DATs, or Digital Audio Tapes, are just now being marketed in the computer industry. They are very similar to 8mm tapes, the main difference being that the medium is 4mm wide instead of 8mm (resulting in a maximum capacity that is half that of 8mm tapes). In addition to being half as thick, DATs are also smaller than 8mm tapes (about 3" by 2").

DAT storage devices use the same tapes as are used by digital audio tape recorders. Commercial DAT recorders for entertainment purposes, however, are currently banned from import due to the intense lobbying efforts of the audio recording industry, which wishes to preserve the current demand for compact disks. These restrictions have effectively reduced the availability of DAT cartridges, so they are generally more expensive than 8mm tapes.

Optical Disks

Optical recording devices were first developed as an alternative to the Video Cassette Recorder. In 1978, the first optical disk system — the Laser-Disk Video Player — appeared on the consumer market. This read-only device used a 12" platter and a laser read-head to play back digitally encoded video signals.

Since then, optical recording technology has been further developed by the mass storage industry, and split into two distinct branches: WORM (Write Once, Read Many) and Erasable optical. A third optical technology, CD-ROM (Compact Disk Read Only Memory), is not commonly used for mass storage since information can only be written during the manufacturing process. CD-ROMs are used largely to distribute and reference large amounts of relatively static data such as on-line encyclopedias, legal citations, and (of course) musical recordings.

WORM and Erasable optical systems use a removable disk enclosed in a plastic cartridge. The main expense involved in optical disk systems is the read/write head, which uses lasers, beam-splitters, lenses, and mirrors to access data. For this reason, most optical systems are single-sided: the disk must be removed from the drive and manually turned over to access the other side of the disk cartridge. An optical disk system that could access both sides of a disk cartridge without removing the disk would require two read/write heads, effectively doubling the cost of the drive.

Jukeboxes, or auto-changers, offer users automated access to a number of optical disk cartridges. Jukebox systems typically contain two optical disk drives, and a mechanical arm used to select and load one of many optical disk cartridges that are stored in the jukebox. These types of systems usually have 5%–10% of their capacity on-line at any one time, and have a maximum capacity approaching one hundred GBytes.

Because the head in an optical disk drive weighs much more than the head in a Winchester disk system, access times for optical drives tend to be much higher than for Winchesters (50–150 ms as opposed to 10–20 ms). Data transfer times, which are dependent solely upon rotational speed, are comparable between the two systems.

The media used in optical disk drives also have numerous advantages over magnetic media. First, since the density of an optical disk is limited only by the wavelength of light used by the laser writing the information, optical disks are capable of tremendous track capacities. Most current systems use a near-infrared laser with a wavelength of 8,000 – 10,000 angstroms, resulting in track densities on the order of 16,000 tpi. A single $5\frac{1}{4}''$ optical disk cartridge with such a track density can hold roughly 800 MBytes of data. Second, the distance between the head and the surface of the disk is much greater than that used by traditional Winchester technologies. A distance of 0.4 mm is typical for optical disks, while Winchesters commonly use 0.0002 mm. This increased separation between the disk and the head makes head crashes very rare. A stray smoke or dust particle, which would cause a head crash in a Winchester disk, will have no affect on an optical disk drive. Finally, the optical disk cartridge itself is very durable. Encased in plastic, it is immune to fingerprints and resistant to heat and humidity.

Optical disk cartridges are expensive in comparison to tape (about $200–$250 each), but their huge capacity makes them cost-competitive with all but helical-scan tape systems.

WORM Optical Drives

WORM optical drives write information by burning small pits in the surface of the disk cartridge, using a laser. Once written, a pit (which represents a binary "1") cannot be restored to a normal flat surface (which represents a binary "0"), so data can be written only once to the same disk sector. To read the data, the same laser is directed at the surface, but at a much lower power setting. The laser is reflected off the surface, and this reflected light is gathered into a photocell. The light reflected by a pit is easily distinguished from light reflected by a flat surface: this difference in light reflections is used to read bit patterns.

WORM optical drives have one major "snag" that is not encountered with other mass storage technologies. Most existing file systems are structured so that some space on the disk is reserved for a directory. This directory must be updated each time a file is added, edited, or deleted. Since WORM optical disks cannot be rewritten, such directory maintenance is impossible. One solution to this problem is to employ special software drivers that use an entirely different file structure involving linked directories. Other solutions involve using a flexible disk to store the directory entries, while the actual data is stored on the optical disk. Directory maintenance limitations, combined with the write-once nature of the media, make WORM optical disk drives useful primarily for backup and archival tasks.

The write-once "limitation" of the medium, however, gives the WORM optical disk drive one unique advantage: once data is written, it cannot be altered. This characteristic makes WORM drives excellent for storing information that must be maintained for legal and audit considerations.

Erasable Optical Drives

Erasable optical disks have only recently been introduced as a viable mass storage technology. Several manufacturers, including Sony, Ricoh, and Hitachi, are now shipping erasable optical systems. A host of smaller companies will be introducing units within the year.

There are three separate technologies associated with erasable optical disks: magneto-optical, dye-polymer, and phase-change. Magneto-optical is the only technology that has reached the production stage, as various problems with the other technologies will require further research before they can be made into marketable products.

Magneto-optical technology, as the name implies, uses a combination of lasers and magnetic field effects to store and retrieve data. The disk is composed of a magnetic material, highly stable at room temperature, encased in a plastic cartridge. To write to the disk, a laser heats a spot on the disk to above 140 degrees Celsius, at which point the magnetic flux can easily be changed by a magnetic head. After the disk cools – only microseconds later – the magnetic flux once again becomes nearly impervious to magnetic fields. It is estimated that, at room temperature, a two ton magnet would be required to change the data on a magneto-optical disk (MOD) cartridge.

The properties of the Kerr effect are used to read data stored on an MOD cartridge. The Kerr effect states that light will rotate in a particular direction if influenced by a magnetic field. An MOD drive uses this effect by directing a low-power laser at the surface of the disk. The light reflected from the surface will rotate in a clockwise or couterclockwise direction, depending upon the orientation of the magnetic flux of the surface. The read head detects the rotation direction, and sends a corresponding value of "0" or "1" to the computer.

Erasable optical disks, however, do have their disadvantages. MOD systems must write zeros to the surface before data may be written to that spot. This means that the disk must rotate twice to complete a write operation: once to write zeros, and once to write the desired information. This quirk effectively increases the write-access time of an MOD drive by 40% over read access times.

The removable MOD cartridge is similar in shape, size, and capacity to the WORM cartridge. A 6" by $5\frac{1}{2}$" by $\frac{1}{2}$" hard plastic casing encloses the disk, and the read/write area is protected by a metal shutter. Unlike WORM disks, MOD cartridges are currently being standardized by both the ISO and ANSI.

Interface Standardization

Manufacturers of interfaces, used for communication between computers and peripheral devices, are just now beginning to recognize the need for standardization. In the past, most large companies used their own interface to connect mass storage devices to their computers. The problem with these interfaces is that they can be used only to connect peripherals to the equipment of a particular manufacturer: forcing manufacturers to equip their peripherals with a different interface for each brand of computer, or limit their market to a small number of computer types.

Computer manufacturers are now beginning to equip their computers with a standard interface, so that any peripheral supporting that standard may be connected. There are currently two major standard interfaces supported by the computer industry: SCSI (Small Computer Systems Interface) and IPI (Intelligent Peripheral Interface).

SCSI, pronounced "scuzzy", is the most widespread standard peripheral interface. Although the term *small* computer systems is used in its acronym, SCSI is widely used for both small and large computer systems.

SCSI uses an 8-bit data path, and has a maximum transfer rate of 2.5 MBytes per second in asynchronous mode, or 5 MBytes per second in synchronous mode. Unfortunately, there is a lot of leeway in the SCSI standard, and many manufacturers have been taking advantage of this. Consequently, a mass storage device that implements SCSI may not always work on every computer system with a SCSI interface.

SCSI is about to be replaced by SCSI–2. This new interface can use a wider data path (up to 32 bits), and has a much faster data transfer rate. SCSI–2 is also downwardly compatible with SCSI.

IPI has been around for only the last five years. It uses a 16-bit data path, and can transfer data at up to 10 MBytes per second. In spite of this seemingly high performance, IPI is not as widely used as SCSI. IPI is used primarily by manufacturers like IBM, and is sold to captive customers rather than to OEMs. IPI is used in fewer than 1% of disk drives sold today.

As more computer manufacturers switch to standard interfaces such as SCSI and IPI, the goal of "plug and play" peripherals will become more of a reality. Hewlett-Packard, for example, has recently announced that their new MOD drives will be equipped with a SCSI interface instead of HP-IB (Hewlett-Packard Interface Bus), which has been a HP staple for years. This is the first step in their effort to move toward a more standardized interface.

Comparing the Technologies

With all technologies, there is some trade-off between speed, capacity, cost, and use. Figure 1 compares the Burst Transfer rates for the technologies discussed so far.

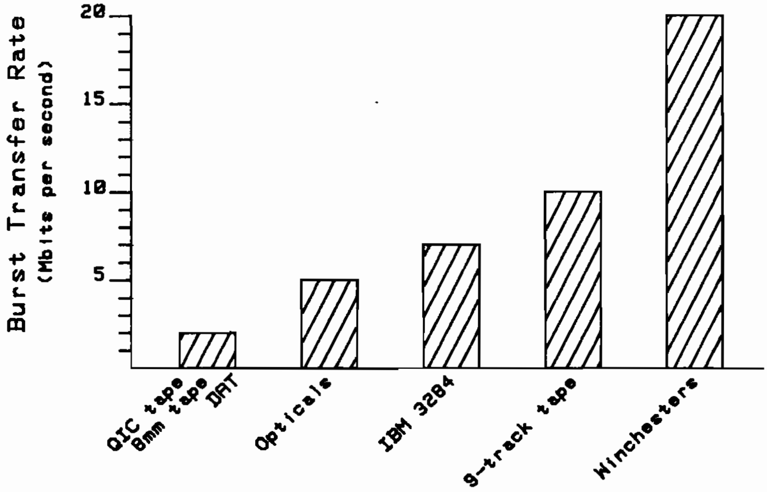


Figure 1: Comparison of Burst Transfer Rates

Figure 2 below shows a range of average access times for Winchesters, Removable disk media, Optical Disks, and Tape drives.

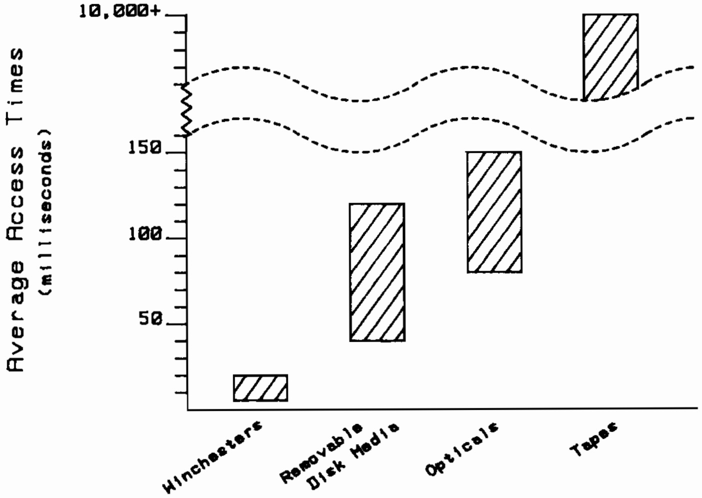


Figure 2: **Comparison of Average Access Times**

Figure 3 presents an analysis of typical storage costs (per MByte of data) for a variety of mediums. The costs shown were figured using the costs of the media only: drive costs were not included.

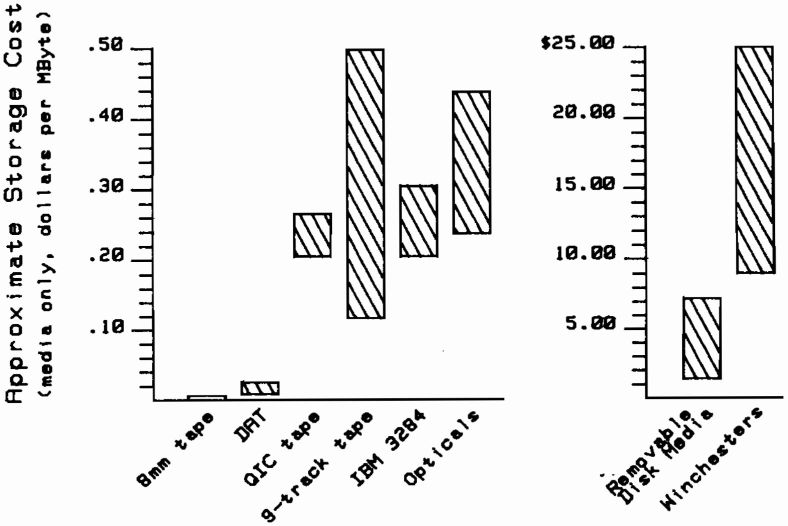


Figure 3: **Cost Comparison for Storage Methods**

Looking to the Future

Helical-scan tape has started the 9-track tape drive on the road to obsolescence. In the coming years, all manufacturers will be using one of a few standard interfaces for their computers, which will make everyone's lives easier. Helical-scan tape, already gaining in popularity, will be the medium of choice for backup and data interchange tasks.

The WORM drive will be supplanted by the more flexible erasable optical disk drive, except for a small niche where data permanence considerations are crucial. Also, the cost of optical media should drop dramatically as more manufacturers enter the market.

A major innovation to look for in the future will be the widespread use of integrated mass storage systems. These systems use a combination of Winchester, optical, and tape technology to store files: where they are stored depends upon the frequency of their use. A particular file will automatically migrate from fast on-line Winchester storage to slower optical and tape systems as the frequency of its use decreases.

Finally, as optical and helical-scan tape systems are refined, their cost, storage capacities, reliability, access times, and transfer rates will all improve.

SCSI: DISK INTERFACE OF CHOICE FOR HP WORKSTATIONS

Jonathan L. Wood
Hewlett-Packard
P.O. Box 39
Boise, ID 83707

WHY SCSI?

The computer industry in general, and the workstation and PC markets in particular, has been moving rapidly to standards-based systems. Standards-based systems have many advantages. A computer user who invests in standards-based hardware is not limited to the product offerings or pricing policy of a single company, and assures that his or her investment is protected. Standards-based products are predicted to make up much of the workstation market in the next few years. Microprocessors, graphical user interfaces, operating systems, networks, disk interfaces, and graphics are all seeing significant standards-based development. As standards in these areas become more defined, shrink-wrapped workstation software will become closer to a reality.

Standards-based hardware is getting much attention, especially in the area of peripherals. Users in the PC market have had plug-and-play hardware compatibility. The standards in the PC arena have been set not by an ANSI committee, but by recognition of de-facto standards. The IBM PC set the standard because it had such a large share of the market. Other computer manufacturers utilized the bus architecture making the PC-AT bus the industry standard architecture (ISA) bus. Many bought disks from Seagate, and soon ST-506 became the de-facto standard interface for disk drives. Microsoft licensed MS-DOS, and it soon became another de-facto standard.

The workstation market has evolved standards in quite a different way. Much of the development of UNIX workstation standards has happened in committees of the American National Standards Institute (ANSI), and in industry-wide committees like X-Open and Open Software Foundation (OSF). The subject of this article, the Small Computer System Interface (SCSI-1) was developed by an ANSI committee, and is defined by ANSI standard X3.T31-1986. SCSI was developed using principles from Shugart Associates System Interface (SASI), which was developed in the mid-seventies.

Hewlett-Packard has been a leader in the formulation and adoption of UNIX workstation standards. This is one of the reasons that the SCSI interface is now standard on the HP 9000 Series 300 workstations. In addition, the HP-IB interface was perceived as a performance limiter though in most cases it was not. Yet another reason for the adoptions of SCSI is that it is well adapted to connect a variety of peripherals, like rewritable optical, Digital Audio Tape, CD-ROM, and printers. The ability to connect all of the peripherals onto one interface eliminates the need for multiple interfaces, thus saving slots for other uses and minimizing cost. With SCSI, users can connect devices that HP currently chooses not to offer, like write once, read many (WORM) optical, and solid-state disk.

The SCSI interface is a high-level interface, as opposed to other interfaces like ST-506, enhanced small disk interface (ESDI), Intelligent Peripheral Interface-2 (IPI-2), and SMD, which are device-level interfaces. Other high-level interfaces include Hewlett-Packard Interface Bus (HP-IB) and Hewlett-Packard Fiber Link (HP-FL), and IPI-3. The major conceptual difference is that the disk controller hides many of the details of disk operation from the host computer. The computer does not have to manage the details of where the file is physically located, and does not have to separate header and trailer information from the data. In addition, the controller presents the disk as one long string of logical blocks, hiding bad sectors. A device-level interface forces the CPU to keep track of bad sectors.

The effect of a high level interface is that the computer is free to do other tasks because the controller handles many details of disk I/O. Single-user, single-tasking machines cannot take advantage of this feature because the computer is not able to go on until the data from the disk arrives. However, a multi-tasking computer can take advantage of the higher availability of the CPU, leading to a significant performance improvement.

Time To Market

One of the features that makes SCSI so attractive to peripheral manufacturers is that an embedded SCSI interface lets them bring a new product to market very quickly. For instance, imagine that XYZ Company develops a new WORM drive. If they do not use an industry standard interface they are caught in a catch 22 situation: nobody will build the controller because the volumes are so low. There are no sales because there is no way to use the product. XYZ Company must convince another company to develop controllers for a variety of computers or do it themselves. Instead, if XYZ company designs their own embedded SCSI controller and puts it on the mechanism, significantly less work is needed to integrate the new product into a computer system. Of course, the computer would also require a SCSI driver compatible with the drive and operating

Table I

SCSI Bus Signals

Signal	Description
ACK	(Acknowledge) Data on bus.
ATN	(Attention) Request for message out phase. Initiator has message for target.
BSY	(Bsy) Target is Busy.
C/D	(Control/Data) Indicates whether bus carries control messages or data.
I/O	(Input/Output) Indicates direction of data flow on the bus. I/O is true when data goes from target to initiator.
MSG	(Message) Signals on the bus are a message when this signal is asserted.
REQ	(Request) Requests data on the bus.
RST	(Reset) Hard reset of all devices on the bus.
SEL	(Selection) Used during selection and reselection.

system. Therefore, peripheral manufacturers can speed their time to market by using the SCSI interface.

SCSI Bus Functionality

In order to understand the complexities of the SCSI bus, including the differences between synchronous and asynchronous data transfer, it is important to have some understanding of the different bus states, signals, and commands. Table I lists SCSI bus signals. Table II lists SCSI bus states.

The commands listed in Appendix 1 are those supported by the HP 9754xS disk that is sold on an OEM basis to other manufacturers. Note that many of the commands are described as "vendor-unique". All SCSI drives support many such commands. The vendor-unique commands are mainly used for diagnostic purposes. Just because a drive uses vendor-unique commands does not mean that it is incompatible with another vendor's hardware. The fact is that the implementation of the SCSI commands of the 9754xS were designed with vendor-independence in mind. In most instances the operating system will never issue a vendor-unique command.

Table II

SCSI Bus States

State	Description
Arbitration	Initiator negotiates for control of the bus.
Bus Free	No device is using the bus. It may be that a target has disconnected and will soon reselect.
Command	Initiator issues a command like read, write, or format.
Message In	Target will disconnect and then the bus will be free.
Message Out	Initiator identifies itself to the target.
Selection	Initiator establishes contact with target.
Status	Target reports on transfer status.
Data In, Out	Data is transferred.

Disk Transaction

The basic unit of disk operation is an I/O. What follows is a description of a typical disk I/O. This example holds true for any SCSI device, not just disks. The I/O operation is initiated by the operating system of the computer, which for this example happens to be HP-UX. Note however, that many different operating systems support SCSI peripherals, including MS-DOS. The disk transaction begins with a request from the computer. The request may be for a file system block or for a virtual memory page. This request is passed to the SCSI driver, along with data that the driver will use to locate the data, including the device identifier. Then the driver goes through an ARBITRATION and SELECTION process to gain control of the bus.

The disk responds with a MESSAGE OUT, and then receives an IDENTIFY message from the host. The disk's response to the IDENTIFY message indicates to the host whether the disk supports disconnect/reconnect during the data phases, and also whether the disk supports command queuing (a SCSI-2 feature). The host driver then asserts the ATN signal to maintain the MESSAGE OUT phase and determines whether the disk can support synchronous transfer. Then the driver issues a command indicating whether it wants a read or a write. Upon receiving this command, the disk sends disconnect and save data pointers to the host, and then disconnects if it supports that option. At this point the bus is free to service other devices. During this time the controller decodes the request and carries it out. When the disk is finished or nearly finished loading data into the buffer, it asserts the RESELECT signal, and transmits data from its buffer. Depending on the size of the disk buffer and the amount of data requested, the disconnect/reconnect cycle may happen several times during an I/O. When the transaction is complete, the STATUS message is sent to the host, and the transfer is complete.

Comparisons

Single Ended vs. Differential

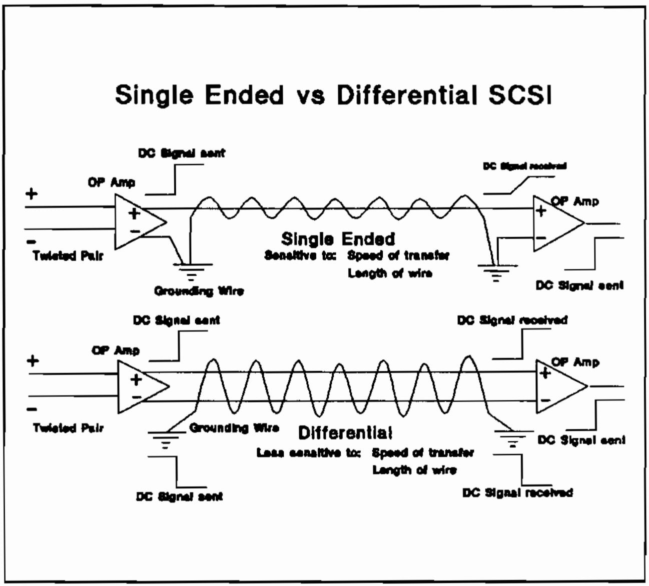


Figure 1

The difference between single-ended and differential SCSI is in the electrical definition of the signal transceivers. Differential SCSI uses twice as many wires as single-ended SCSI. In single-ended, one wire of a pair is used for grounding only, whereas differential SCSI uses the second wire to send the complement of the signal in the first wire. In other words, as the first line goes high, the second line goes low. This has the effect of decreasing electromagnetic emissions because the fields generated by the two wires tend to cancel each other out, like a coaxial cable does. This gives a higher signal to noise ratio, which allows faster clock speeds and longer cable lengths. See Figure 1.

Synchronous vs. Asynchronous

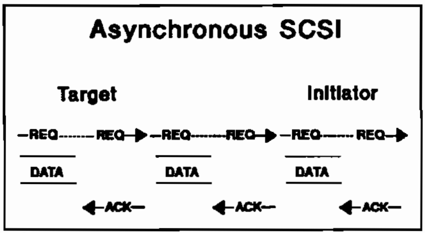


Figure 2

Synchronous SCSI-1 is rated at 5 Mbytes per second, whereas asynchronous is rated at only about 1.5 to 3 Mbytes per second. Note that all commands and messages are transmitted at the asynchronous rate. Only data can be processed in synchronous mode. When a transfer is initiated by the host computer, either device can issue a "synchronous data transfer request" (SDTR) message. If either the initiator or target fails to issue the message, the transfer defaults to asynchronous. Asynchronous means that for each 1 byte transfer, the target sends an ACK signal and then waits for an REQ signal from the initiator before more data is sent.

Thus, the ACK/REQ handshake must happen for each byte of data transferred in asynchronous mode. During a synchronous transmission, the target device does not wait for the REQ signal between data transmissions. Instead, data is sent until a number of bytes, determined by the REQ/ACK offset agreed upon at the initiation of synchronous transfer, has been sent. During the transmission, the initiator continues to send ACK's back to the target. The target keeps track of them and knows that when the number of REQ's matches the number of ACK's the transfer is complete. See Figures 2 and 3 for a graphical description.

SCSI-1 vs. SCSI-2

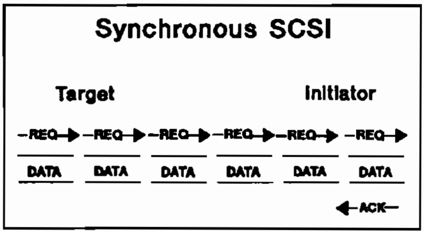


Figure 3

The SCSI-1 interface is rated at up to 5 Mbytes per second for synchronous data transmissions. The SCSI-2 definition allows for data transmission at up to 10 megatransfers[1] per second (differential cables only), giving differential SCSI-2 data transfer rates up to 320 megabits, or 40 Mbytes per second. There are two different kinds of modifications to the SCSI-1 standard that allow the higher data rate.

The first modification is called "fast". Fast SCSI-2 requires differential electronic drivers. Single-ended electronics are not capable of the signal to noise ratio necessary to implement

[1] As will be explained, each transfer can be 8, 16, or 32 bits wide.

fast SCSI-2. The clock speed of fast SCSI-2 is twice that of SCSI-1. Using the same 8 bit data path, fast SCSI-2 is capable of 10 Mbytes per second. The cable length limit of differential SCSI is 15 meters compared to 6 for single-ended.

The second modification affecting transfer rate is "wide" SCSI. Wide SCSI-2 gives a data path of either 16 or 32 bits as opposed to the 8-bit path of SCSI-1. The space required to connect the cables may preclude the use of wide SCSI on 2.5" and 3.5" disks. By combining wide and fast, SCSI-2 can achieve a synchronous data transfer rate of 40 Mbytes per second.

Implications of SCSI-2

Most applications will not need a wide and fast SCSI-2 bus. Doubling the transfer rate of the bus will not double the throughput of the I/O system. In a random disk transfer, the seek and latency constitute a far larger chunk of time than the channel time. An average 5.25" disk has an average seek plus latency of 24 milliseconds compared to less than 4 milliseconds for an 8 KByte transfer. Even a 40 Mbyte per second channel would not necessarily decrease the 4 millisecond figure because it takes a substantial amount of time to get the data from the disk surface once the heads are over the data.

For a standalone workstation, fast SCSI or even synchronous SCSI-1 provides more bandwidth than a single or dual disk drive configuration can utilize. Very few 5.25" disks used on workstations have a sustained UNIX file system transfer rate of greater than 2.0 Mbytes per second, even for large files.

Certain applications, however, could certainly benefit from a wide and fast SCSI-2 bus. Examples include file servers with multiple disks, transaction processing again with multiple disks, solid state disks, disk arrays, and processor-to-processor communication.

SCSI-2 has other features that differentiate it from SCSI-1 besides transfer rates. SCSI-1 allows only one outstanding command from an initiator to a target. Command queuing in SCSI-2 allows the host I/O driver to handle multiple requests. Command queuing allows up to 256 requests to be outstanding from each initiator to each target. SCSI-2 has much more tightly defined electrical specifications. SCSI-1 electrical specifications are loose enough that two devices that both meet the specification could not work together. SCSI-2 closes the gaps.

SCSI-1 vs. HP-IB

The two interfaces currently available on The HP 9000 Series 300 workstations are HP-IB and SCSI-1. HP-IB is an interface based on IEEE-488 and is rated at a transfer rate of 1 Mbyte per second. The Hewlett-Packard disks available for Series 300 workstations include the Series 6000 Model 670H and Model 660S. The Model 670H uses HP-IB, while the Model 660S uses SCSI-1. In order to characterize the performance differences between these two disk drives, several benchmarks were executed.

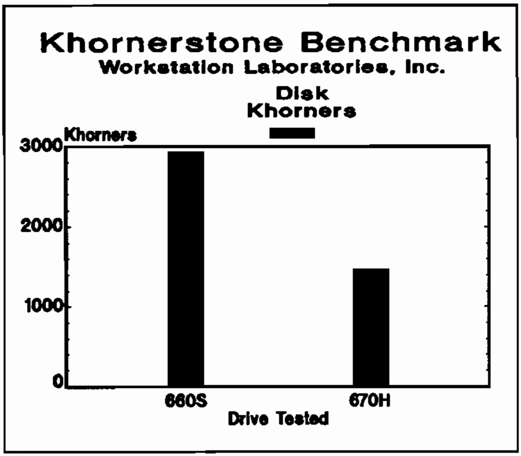


Figure 4

The Khornerstone benchmark is owned by Workstation Labs, Inc, and the disk portion of the test includes disk intensive tasks, such as reading and writing files both randomly and sequentially. As can be seen in Figure 4 the Model 660S has a disk Khornerstone score about twice that of the Model 670H. Figures 5 and 6 show the throughput of these two disks for files of various sizes, both reading and copying (reading and writing).

The Model 660S and the Model 670H share the same disk mechanism and ESDI device-level interface. The differences are:

- The firmware of the Model 660S has been tuned for HP-UX.
- The SCSI channel is much faster than HP-IB.

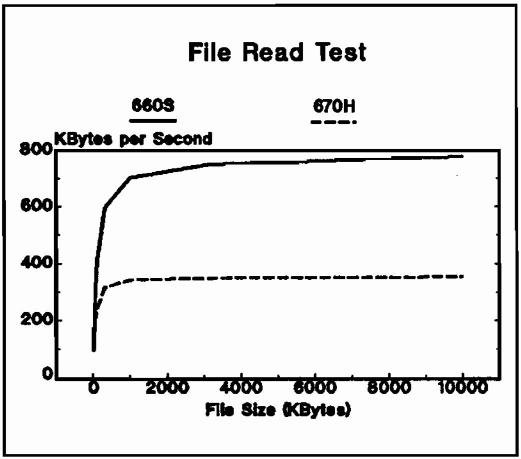


Figure 5

The net effect of these two changes is dramatic. Whether the tuning of the Model 660S' firmware or the faster channel makes the biggest difference, the performance choice for HP Series 300 workstations is clearly SCSI.

SCSI vs. ESDI

As previously stated, ESDI is a device-level interface. Many SCSI disk drives use ESDI as the device-level interface under SCSI. The difference between SCSI and ESDI is that whereas ESDI defines a device controller, SCSI defines an interface bus.

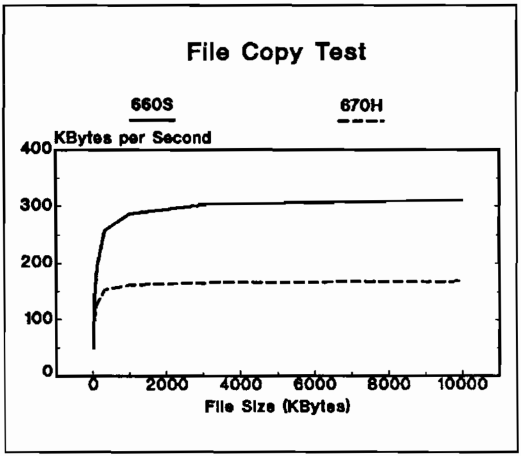


Figure 6

Because SCSI has more intelligence than ESDI, it has more overhead. SCSI does some of the work that the central processor would do for an ESDI device. In some environments moving intelligence from the CPU to the controller does little good. For example, PC's often perform better with an ESDI drive than with a SCSI drive, at least in a single drive configuration running a single-tasking operating system (DOS). The reason is that the added overhead of decoding the SCSI commands has a greater effect than the higher availability of the CPU. In the DOS environment the CPU cannot work on other tasks in the background. However, when a computer runs a multi-tasking operating system and uses multiple drives, SCSI will outperform ESDI.

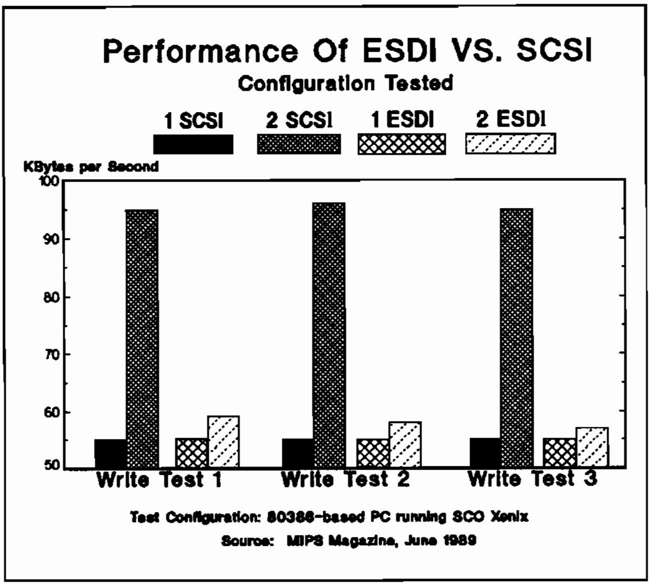
The SCSI standard allows a device to disconnect from the bus during long operations such as formatting, seeking, and tape positioning. The added availability of the SCSI bus as a result of the disconnect/reconnect feature gives SCSI an advantage over ESDI. Again, the difference between the two is small for single drive configurations. However, when multiple devices share the same bus, the throughput of the SCSI system is much greater than for the ESDI system.

Figure 7 shows the results of several sequential write benchmarks that were executed on an 80386-based PC. The significance of the slide is not the relative performance of the SCSI and ESDI drives tested. The significance of this data is what happens as more disks are added. Notice that the throughput of the SCSI bus increases dramatically as more disks are added. The ESDI channel also shows some improvement, but not nearly as much as the SCSI channel.

In choosing between an ESDI interface and a SCSI interface, the main questions are these:
1) Will the computer be used in a single-user-single-task, single-user-multiple-task, or multiple-user-multiple-task environment? As more processes and users access the disk(s), SCSI will give better performance than ESDI.

2) Will other types of peripherals be used? Many different types of devices are available that have an embedded SCSI interface. It is possible to have an ESDI interface on a tape

drive, but none are available. On the other hand, many different vendors offer SCSI peripherals, including 8 MM tape, 4 MM tape, 1/4" tape, plotters, CD-ROM, rewritable optical, write-once-read-many (WORM), solid-state disk, and even printers.



Test Configuration: 80386-based PC running SCO Xenix

Source: MIPS Magazine, June 1989

Figure 7

SCSI vs. IPI

One of the main differences between SCSI and IPI is that while SCSI is a peer-to-peer bus, IPI is a master-slave bus. The IPI interface is really a hierarchical set of interfaces. IPI-0 defines a 16 bit parallel cable, IPI-1 defines a communication protocol. Together IPI-1 and IPI-2 define a device-level interface with nearly the same characteristics as ESDI and the SCSI hardware specification. IPI-3 defines a high level interface similar to full SCSI. Most of the IPI systems sold connect to IBM mainframes.

IPI-2 has a transfer rate of 10 Mbytes per second, the same as fast SCSI-2, making it a good choice where fast I/O is needed. One of the drawbacks of using IPI-2 or -3 on a workstation is that IPI disks tend to be much more expensive than SCSI disks because almost all of the IPI disks available are 8" or larger and are dual ported (they have two read and write heads).

The IPI interface is best suited for large servers and mainframes. The high cost may be justified by very high throughput 8 inch disks. However, multiple smaller disks on a SCSI-2 interface may give better performance at a lower cost per Mbyte, especially in a random server environment, where I/O's per second are more important than Mbytes per second. The larger number of spindles would allow concurrent seeks, which are very important to performance in a random multi-user environment.

Summary

Standards are becoming more and more pervasive. The SCSI interface is becoming the standard peripheral interface for Unix-based workstations. Many manufacturers produce mechanisms with embedded SCSI controllers, and as the SCSI standard becomes more tightly defined, incompatibilities among SCSI devices will decrease. SCSI is ideally suited to the workstation environment for these reasons:

- Many different types of devices are being produced with an embedded SCSI controller. Users who have a SCSI interface will be able to make use of them.

- The SCSI bus has high throughput. As wide and fast SCSI products become available, the performance of multi-disk systems can be expected to improve.

- SCSI is better adapted to a multi-user environment than ESDI, yet is more cost effective than IPI.

- By using embedded SCSI controllers, manufacturers can reduce the amount of integration work needed to bring a product to market.

Appendix 1.

SCSI Commands Supported By The HP 9754xS Disk

COMMAND DESCRIPTION

ACCESS LOG
Vendor-unique command. Requests Target to retrieve information from its maintenance log.

EXECUTE DATA
Vendor-unique command. Executes special code downloaded via the WRITE BUFFER command.

FORMAT UNIT
Formats Target media into Initiator addressable logic blocks. Defect sources include P, D, and G lists (no C list). When formatting, it is recommended that the Initiator not include a D list (FMTDAT=0). However, if the Initiator does include a D list, it must be in the physical sector or bytes from index format. The Target uses an interleave of 1 regardless of the value in Interleave field.

INQUIRY
Requests Target to send parameter information to the Initiator. Additional Vital Product Data (VPD) may be supplied if requested by the Initiator.

INTERFACE CONTROL
Vendor-unique command. Enables ESDI commands to be sent to the disk drive processor.

MANAGE PRIMARY
Vendor-unique command. Used to manage the primary defect list (P list). This command can delete the current P list, install a new P list, or append defects to the current P list.

MEDIA TEST
Vendor-unique command. Used to test the integrity of the disk media.

MODE SELECT
Enables Initiator to specify media, logical unit, or device parameters to the Target.

MODE SENSE
Enables Target to report its media, logical unit, or device parameters to Initiator.

READ
(6-byte)
Requests Target to transfer data to Initiator.
Relative Addressing not supported in extended

(10-byte)
(10-byte) format (REL=0).

READ BUFFER
Used with WRITE BUFFER command to test the Target's data buffer. Recommend executing RESERVE command to guarantee data integrity.

READ CAPACITY
Enables Initiators to request information regarding the capacity of a logical unit. Use of PMI bit supported. Relative Addressing not supported (REL=0).

READ DEFECT DATA
Requests Target to transfer media defect data to Initiator. Target returns P, G, or P+G lists in physical sector or bytes from index format.

READ HEADERS
Vendor-unique command. Requests Target to read all the headers on the addressed track and return the requested number of bytes of header information.

READ FULL
Vendor-unique command. Requests Target to return the header, data field and ECC bytes of one physical sector.

REASSIGN BLOCKS
Requests Target to reassign defective logical block to an area on logical unit reserved for this purpose. It is recommended that the defect list contain only one defect location per command.

REFORMAT TRACK
Vendor-unique command. Formats a single track. If HS bit is 0, then it uses normal default header information. If the HS bit is 1, the supplied header information is used for the track logical address and flag bytes.

RELEASE
Release previously reserved logical units. Third-Party Release supported. Extent Release not supported.

REQUEST SENSE
Requests Target to transfer sense data to the Initiator, including: Sense Key (0-6,B,E), Additional Sense Code, Device Errors (DERRORS), and Recommended Actions. The Bit Pointer and Field Pointer fields are not used. Only the Extended Sense Data format is supported.

RESERVE
Reserves logical units for use of Initiator. Unit and Third-Party Reservations are supported. Extent Reservation are not supported.

REZERO UNIT

Requests Target to perform a recalibrate and then to seek to logical address 0.

SEEK
(6-byte)
(10-byte)

Requests Target to seek to a specified address.
Target returns GOOD status when seek is complete.

SEND DIAGNOSTIC

Requests Target to perform specified diagnostic tests. Self-test is supported. If self-test fails, Check Condition status indicates that results are available via REQUEST SENSE command.

SPECIAL SEEK

Vendor-unique command. Requests Target to leave the disk drive selected after execution of a seek. Allows for special testing at the seek address.

START/STOP UNIT

Requests Target to enable or disable the logical unit for further operations. Using the immediate bit on START is supported, but not recommended.

TEST UNIT READY

Checks Target spindle for proper speed. Target returns GOOD status if drive is up to speed.

VERIFY

Requests Target to verify the data written on the media by performing a selectable ECC check or a byte compare. Relative addressing not supported. (REL=0).

WRITE
(6-byte)
(10-byte)

Requests Target to write the data transferred by the Initiator to the media. Relative Addressing supported in 6-byte format. Relative Addressing not supported in extended (10-byte) format (REL=0).

WRITE AND VERIFY

Requests Target to write the data transferred by the Initiator to the media, then do an ECC verify of the data that was written. Relative addressing not supported. (REL=0, BYTCK=0).

WRITE BUFFER

Used to test Target's data buffer or download code. To avoid possible data corruption, it is recommended that a RESERVE command be executed prior to the WRITE BUFFER command.

WRITE FULL Vendor-unique command. Requests Target to write one complete physical sector, including header, data, and ECC fields.

Linking Voice, HP NewWave and HP OpenMail

Tony Jones
Hewlett-Packard Co.
2 Choke Cherry Road
Rockville, MD 20850

Abstract

NewWave provides the capability for users to maintain their current investment in MS-DOS applications, while providing an integrated application environment. As an example of this integration, this paper will describe a NewWave encapsulation of a commercially available MS-DOS voice product. In addition, this paper will describe how to use HP OpenMail to send voice over multi-vendor networks.

In my implementation, the voice product appears as two icons in the NewWave environment:

Icon 1 features (NewWave data object)

- Voice messages: can be mailed via electronic mail, copied, and archived.

Icon 2 features (NewWave tool)

- Dictation with voice cut and paste
- Phone book with voice message and telephone number dialing
- Answering machine with Message forwarding
- Calendar with Voice message on alarm

The presentation will include a live demonstration of Voice Integration.

Introduction to the NewWave Environment

The consistent and predictable graphical user interface makes the environment easy to learn and easy to use. Users can focus on results, not the tools used to accomplish them. The environment is based on Microsoft Windows/286.

Those working in the NewWave environment can move quickly and easily from one application to another because of the high level of integration. NewWave primarily consists of two components: the Object Management Facility (OMF) and the Application Program Interface (API). Data files and the executable programs which interpret the data are called objects. When the user accesses the object, the OMF starts the application and passes it the appropriate data file. Figure 1 is an example of the NewWave environment.

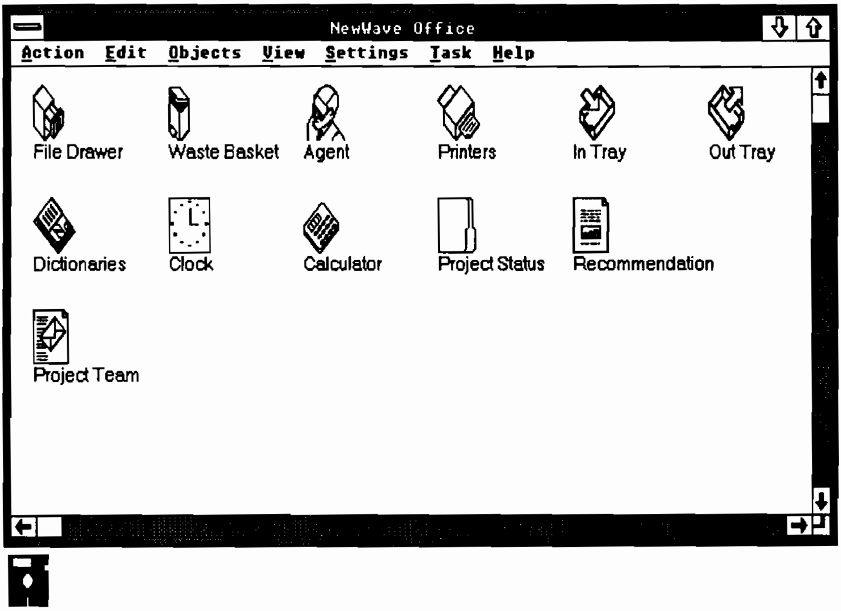
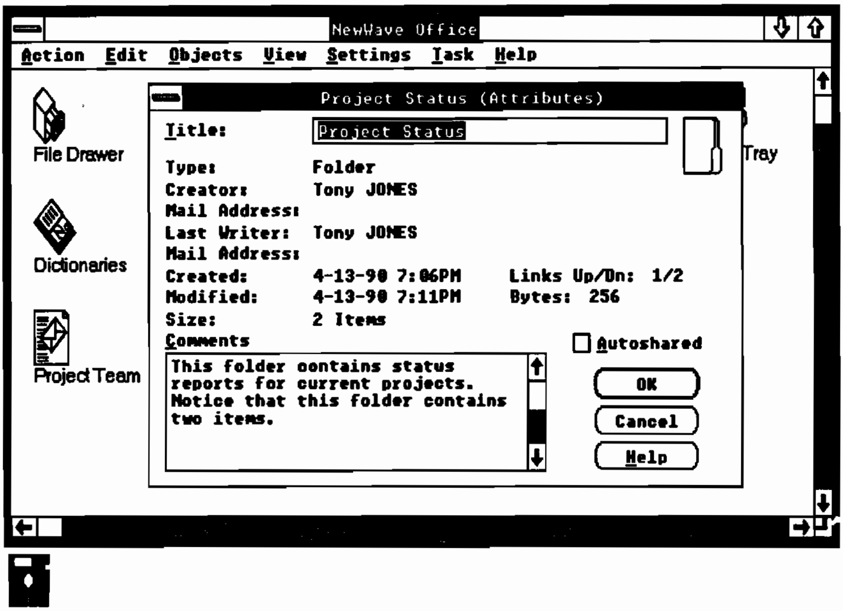


Figure 1: NewWave Environment

Objects are represented as icons. Each NewWave object has a title which is specified by the user. These titles can be up to 32 characters, to provide a way to more clearly describe the item. This is in contrast to the typical PC environment where the user must fit a description into an 8 character file name. In addition, NewWave maintains a set of attributes for each object. These attributes include the creator, type of object, creation date, last modification date, and a comments area. Figure 2 is an example of the attributes area.



Project Status (Attributes)

Title: Project Status

Type: Folder
Creator: Tony JONES
Mail Address:
Last Writer: Tony JONES
Mail Address:
Created: 4-13-90 7:06PM Links Up/Dn: 1/2
Modified: 4-13-90 7:11PM Bytes: 256
Size: 2 Items
Comments ☐ Autoshared

This folder oontains status reports for current projects. Notice that this folder contains two items.

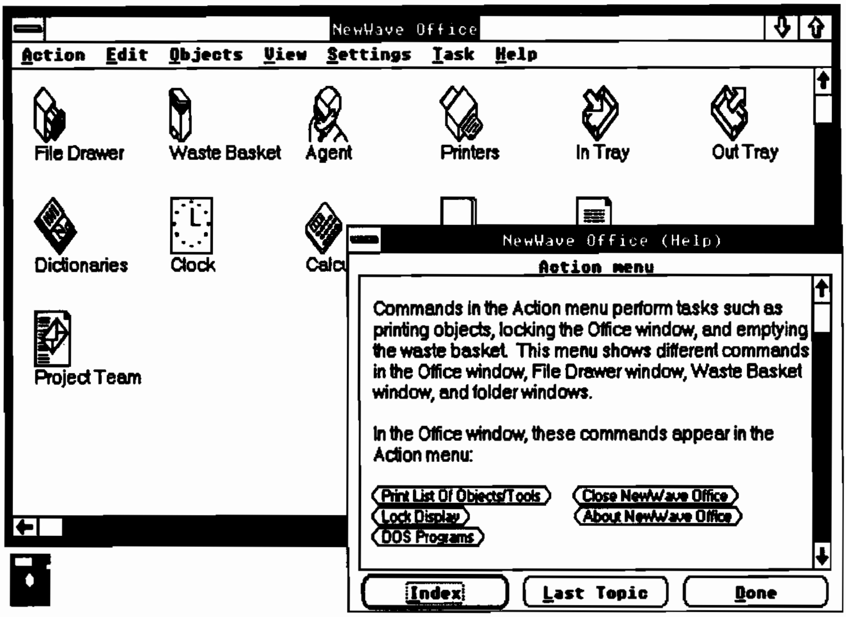
Figure 2: Object Attributes

Objects can be combined with other objects to create compound documents. The OMF keeps track of object relationships through information links. A user can link an object to multiple objects so they "share" the same information. If the information changes in the source object, the OMF notifies and updates all dependent objects. The user is not burdened with finding the files which need to be updated.

NewWave also allows objects to pass data to other objects through a data passing link. An object can focus on retrieving information and depend on a graphics object to plot the information through the use of a data passing link. Visual links allow an application to provide a view of another object's data.

Application developers can take advantage of existing NewWave applications and reduce re-creation of existing software. Since the OMF provides a consistent architecture for data sharing among applications, newly developed applications will work with current NewWave applications. To facilitate development further, NewWave provides the Application Program Interface (API).

The API provides access to system wide services: context sensitive help (Figure 3), task automation by the Agent, and computer based training. The context sensitive help system is a complete facility for developing and displaying help messages. There are three methods for obtaining help: index, inputting a topic, and an on screen pointer (question mark). The index can be used to scroll through the available help topics. If the user chooses to input a topic, the help facility jumps to the matching topic as the user enters each character. The third method is a context sensitive question mark where the user can place the pointer on a specific item in question. Developers can also take advantage of the hypertext-like facilities to provide links to related help topics. A related topic is designated by a group of words circled in the help text.



Commands in the Action menu perform tasks such as printing objects, locking the Office window, and emptying the waste basket. This menu shows different commands in the Office window, File Drawer window, Waste Basket window, and folder windows.

In the Office window, these commands appear in the Action menu:

Figure 3: Context Sensitive Help

System wide task automation is provided by the agent. User actions are translated into commands and recorded in a command file called an agent task. The agent can be triggered by an event or time, by an application call, or by user request. Developers can provide agent tasks which automate routine user activities. Since the API can monitor and control applications, NewWave builds on this to provide Computer Based Training (CBT).

CBT enables developers to provide interactive training to application users. Developers can design one application and have it operate as usual or have it controlled by a lesson. Furthermore, the lesson developer can work without knowledge of Microsoft Windows or NewWave programming. The course developer and application developer can work in parallel.

The Native Language Support (NLS) facility enables developers to provide applications which adapt to languages and customs of other countries. This means native language applications can be produced and maintained with less effort.

Finally, NewWave provides facilities for integrating current MS-DOS and Windows based applications into the NewWave environment. This technique is called encapsulation. Encapsulation refers to the process of building a software shell around an existing application that allows it to achieve higher levels of NewWave integration. Each level requires increased development effort (see Figure 4). Encapsulation is handled by the Bridge Builder.

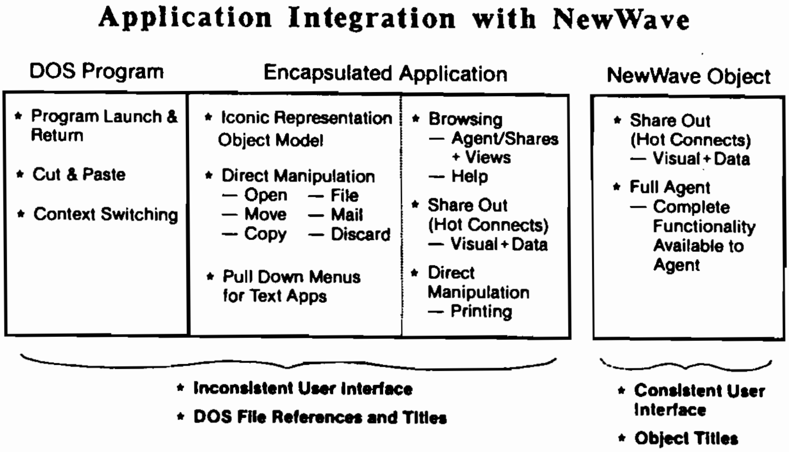

Application Integration with NewWave
DOS Program
Encapsulated Application
NewWave Object
* Program Launch & Return
* Cut & Paste
* Context Switching
* Iconic Representation Object Model
* Direct Manipulation
— Open — File
— Move — Mail
— Copy — Discard
* Pull Down Menus for Text Apps
* Browsing
— Agent/Shares + Views
— Help
* Share Out (Hot Connects)
— Visual + Data
* Direct Manipulation
— Printing
* Share Out (Hot Connects)
— Visual + Data
* Full Agent
— Complete Functionality Available to Agent
* Inconsistent User Interface
* DOS File References and Titles
* Consistent User Interface
* Object Titles

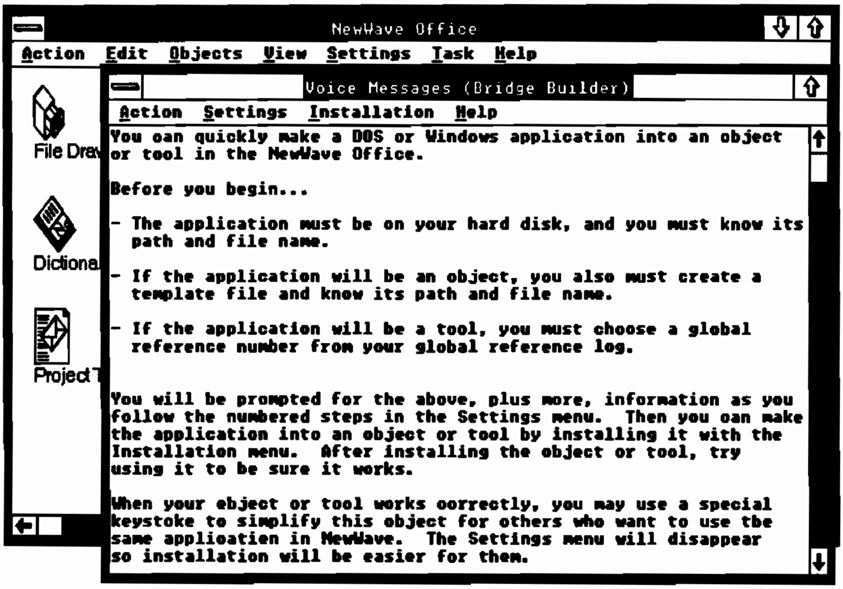
Figure 4: Levels of NewWave Integration

The Bridge Builder

The Bridge Builder is an object which assists in connecting DOS and Windows based applications to NewWave. Once a bridge is defined and installed, an application can appear as an icon (see figure 5). In addition, NewWave can manage the location of files associated with that particular object. DOS or Windows applications can appear as DOS objects or DOS tools. Each DOS object has up to an eight character title and associated data file(s). When a user opens the object, NewWave starts the application and passes it the file name(s) such that the user can immediately work with the information. Users can mail, copy, move, archive, share or delete objects.

DOS tools provide functions which are usable throughout the environment. For example, the Waste Basket tool provides a means for users to throw away objects. A user can only have one instance of any particular tool. In addition, tools cannot be copied or deleted. Bridges allow users to treat DOS or Windows applications as objects or tools.

The Bridge builder also provides facilities for adding pull down menus to DOS (character based) objects or tools. Windows based applications do not require this facility because they should provide menus which adhere to the Windows user interface design rules.


NewWave Office
Action Edit Objects View Settings Task Help
Voice Messages (Bridge Builder)
Action Settings Installation Help
File Drav
Dictiona
Project T

You oan quickly make a DOS or Windows application into an object or tool in the NewWave Office.

Before you begin...

- The application must be on your hard disk, and you must know its path and file name.

- If the application will be an object, you also must create a template file and know its path and file name.

- If the application will be a tool, you must choose a global reference number from your global reference log.

You will be prompted for the above, plus more, information as you follow the numbered steps in the Settings menu. Then you oan make the application into an object or tool by installing it with the Installation menu. After installing the object or tool, try using it to be sure it works.

When your ebject or tool works oorrectly, you may use a special keystoke to simplify this object for others who want to use tbe same applioatien in NewWave. The Settings menu will disappear so installation will be easier for them.

Figure 5: Bridge Builder

Watson Voice Product

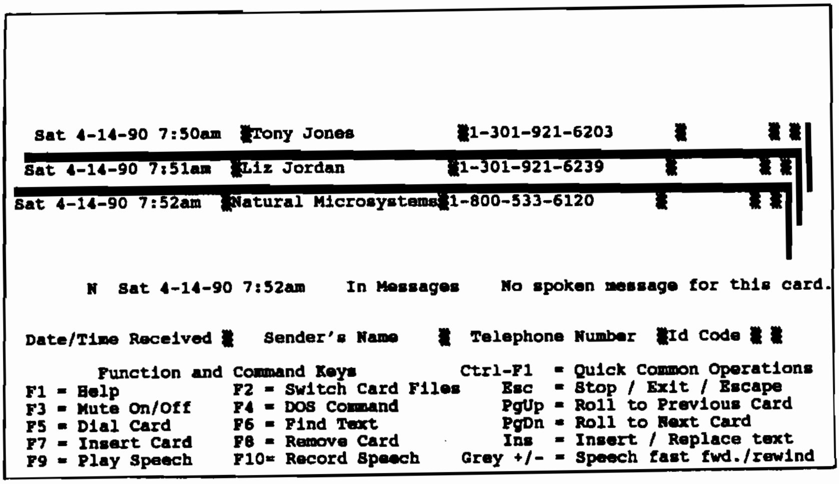
Natural Microsystems Corporation produces an MS-DOS based voice product called Watson. Watson is a voice messaging system that includes either a 1200 or 2400 baud modem. The most common methods for using Watson include:

- Personal Voice Messaging. Watson manages incoming and outgoing messages similar to an answering machine, provides dictation capability with voice editing, provides phone book and telephone number dialing, and a personal calendar.

- Multi-User Voice Mail. A group of people can use Watson to send and receive messages.

- Modem. Watson provides 1200 or 2400 baud modem support which is accessible by data communication applications.

Watson uses card files to manage information, for example placing each incoming or outgoing message in a separate card. Five card files are included with Watson: incoming messages, outgoing messages, phone book, dictation, and calendar. Figure 6 is an example of an incoming messages card file.

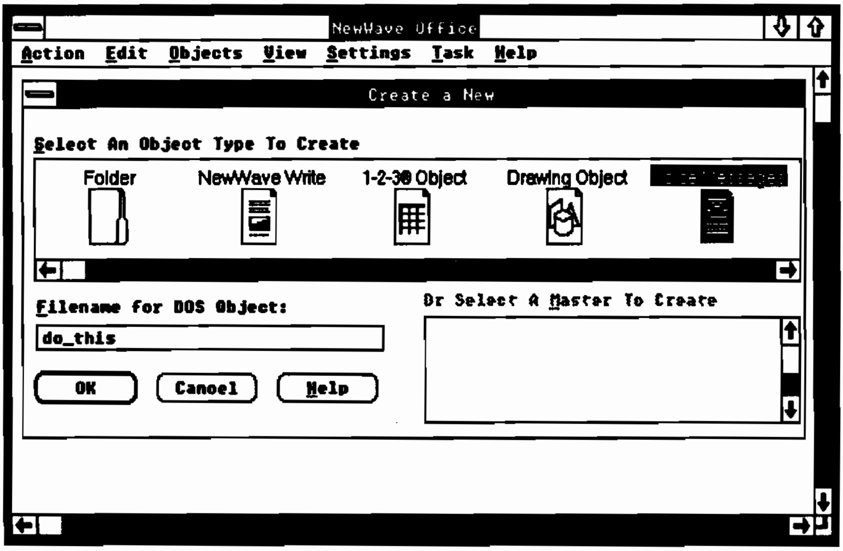


```
 Sat 4-14-90 7:50am ▓Tony Jones          ▓1-301-921-6203     ▓         ▓ ▓
Sat 4-14-90 7:51am ▓Liz Jordan          ▓1-301-921-6239     ▓         ▓ ▓
Sat 4-14-90 7:52am ▓Natural Microsystems▓1-800-533-6120     ▓         ▓ ▓


     N  Sat 4-14-90 7:52am    In Messages    No spoken message for this card.

Date/Time Received ▓   Sender's Name     ▓  Telephone Number  ▓Id Code ▓ ▓

       Function and Command Keys           Ctrl-F1  = Quick Common Operations
F1 = Help             F2 = Switch Card Files   Esc  = Stop / Exit / Escape
F3 = Mute On/Off      F4 = DOS Command         PgUp = Roll to Previous Card
F5 = Dial Card        F6 = Find Text           PgDn = Roll to Next Card
F7 = Insert Card      F8 = Remove Card         Ins  = Insert / Replace text
F9 = Play Speech      F10= Record Speech   Grey +/- = Speech fast fwd./rewind
```

Figure 6: Watson Voice Product

Using the Voice Object/Tool

At this point, Watson appears as two icons in NewWave. Icon 1 represents the phone tool and icon 2 represents voice objects. Managing voice objects is very similar to managing other NewWave objects. To create a voice message, select "Create a New..." from the Objects menu, choose a voice message object from the palette of icons and give it a title.


NewWave Office
Action Edit Objects View Settings Task Help
Create a New
Select An Object Type To Create
Folder
NewWave Write
1-2-3® Object
Drawing Object
Or Select A Master To Create
OK
Cancel
Help

Filename for DOS Object: do_this

Figure 7: Create a New Object

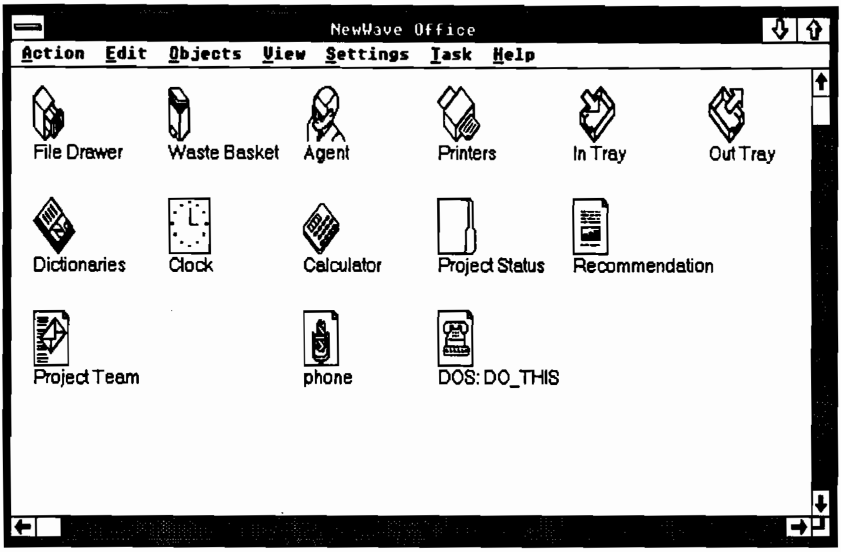
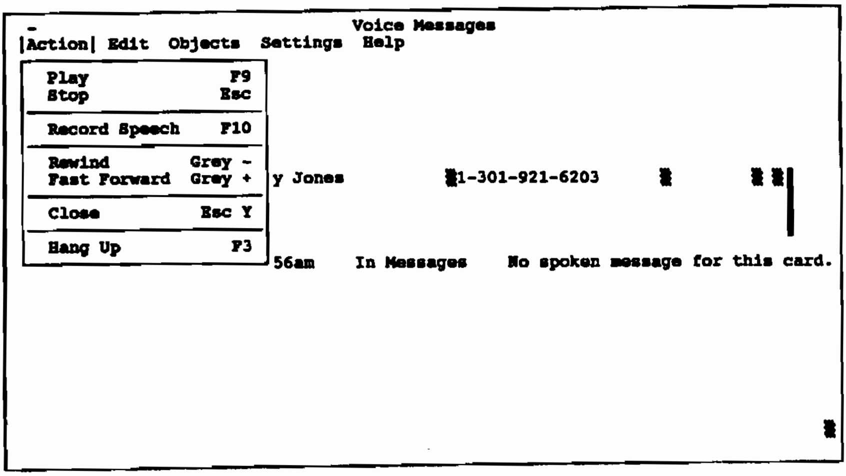


Figure 8: Voice Tool and new Voice Object

To record a message, open the object, press the 'ALT' key to get the menu and select "Record Speech" from the Action menu; pick up the phone and speak (figure 9). When you are through with your message, hang up the phone. Each message has a text area available for comments. In addition, each voice message object can contain a complete set of Watson card files. To return to NewWave, press 'ALT' and choose "Close" from the Action menu. The standard Watson commands are still available for use.



y Jones 1-301-921-6203

56am In Messages No spoken message for this card.

Figure 9: Voice Object with pull down menus

The phone tool provides typical Watson functionality. For example, to use the personal computer as an answering machine or phone book with dialer, open the phone tool. Adding or deleting phone cards in the phone tool has no effect on voice objects.

NewWave allows users to maintain their investment in current applications while providing an environment that assists in making more consistent application usage.

HP OpenMail

OpenMail provides information distribution services. Users access OpenMail via PCs, terminals, UNIX workstations, Apple Macintoshes and can communicate with each other through local or multi-vendor mailing systems. In addition, OpenMail supports transmission of multi-media data types such as NewWave objects.

OpenMail uses various communication methods for exchanging mail: X.400, SendMail, and HP DeskManager. X.400 is popular over wide area networks. SendMail is a defacto standard transport mechanism for UNIX based computers, and DeskManager is the mailing system available for HP 3000 computers which can provide gateway services to IBM mail systems and HP Telex. HP NewWave Mail will integrate with OpenMail to provide a transparent method for exchanging NewWave objects and text messages.

Sending Voice via OpenMail

To mail a voice message, a user places the voice object on the Out Tray. The Out Tray prompts the user for a list of recipients. Distribution lists provide a convenient method for storing user names and addresses. Once the user enters names or distribution lists, NewWave Mail sends the object to OpenMail which in turn delivers the object. OpenMail sends objects over multi-vendor wide area or local area networks.

Conclusion

It should be emphasized that NewWave provides for integration across applications in addition to task automation. Users and developers alike can create new and exciting solutions by integrating feature sets of current DOS, NewWave and future NewWave applications.

References

Windows screen snapshots generated with "Tiffany" program by Alan Anderson, San Diego, CA.

UNIX is a trademark of AT&T.

Microsoft, MS-DOS and Windows are trademarks of Microsoft Corporation.

Apple, Macintosh are trademarks of Apple Corporation.

Are IPC And RPC Enough For System Integration?

Thong Pham

Industrial Applications Center
Hewlett-Packard Company
1266 Kifer Road
Sunnyvale, California 94086

Abstract

Creating integrated systems for factory control and for other purposes using software applications as building blocks requires the ability to send messages between applications. Inter-process communication (IPC) and Remote Procedure Calls (RPC) are usually suggested as tools to message between software applications. In this paper, IPC and RPC will be evaluated to determine whether they are sufficient to integrate software applications.

Requirements of software application will be discussed. Problems not solved directly by IPC and/or RPC will be described in detail. These problems include achieving location transparency, varying syntax with heterogeneous systems, resource management, and achieving reliability and recoverability. A new tool to solve these problems will be presented.

1. Introduction

Over the years we have heard rhetoric about achieving system integration with network services, with inter-process communication (IPC) or with remote procedure calls (RPC). Are these facilities sufficient for the integration of software applications? It is the objective of this paper to answer this question.

First, we will define application integration, focusing on requirements. Due to pressing business operation needs and costly implementations of integrated systems, people have resigned themselves to accepting half-way solutions. This usually results in faulty integrated systems, systems that require a lot of human intervention to keep them functioning and a lot of continuous adjustment. These systems are unreliable because they break down so often. They are hard to change and maintain because they were hard coded. They became a hindrance to the business operation rather than a help. In this paper we will define a truly integrated system, not just an idealized one that only exists in one's dreams but one that can be built with a realistic and useful software tool.

Next, inter-process communication (IPC) and remote procedure call (RPC) mechanisms will be examined. Although IPC has been a reality now for more than a decade, and no one can deny its critical function in a network application, it has not brought any substantial progress to the software integration process. We will determine why that is so as we look at integration requirements that are met or not met by IPC systems. RPC, on the other hand, came into existence around five years ago, and in recent years has gained more and more recognition as an essential tool for the building of distributed systems. Is RPC the right software tool for application integration? We will answer this question by examining application integration requirements that are met by RPC and those that are not.

Last, we will describe a software tool that can address most of the application integration needs.

2. Software Applications Integration

People usually define application integration by what they perceive achievable within their system and programming limits. This restricts their definition of application integration to something less than what actually is possible. For example, most of us are used to the idea of integrating applications by sending files from one system to another. We have built an integrated system using network file transfer services and have failed to admit that we have ignored the many benefits a message-based integrated system can offer.

Let us first think of a truly desirable integrated system, identify its requirements and then try to see what we need to do to meet them. According to Glenn Graham of Coopers and Lybrand (*Automation Encyclopedia, A to Z in Advanced Manufacturing*), "Integration means that the information required by each activity is available on a timely basis, accurately, in the format required, and without asking." Applications integration requires more than just the sharing of data. It requires synchronized actions between participant applications, and if possible, a coordinated presentation of data to the users. Software integration should result in a system that is an integrated whole, not a collection of independent components. To achieve this level of integration, an integrated system must:

- **deliver the right data, at the right time, in the right format, automatically and reliably.**

  An application can only function correctly if it is provided with accurate data in a timely manner. An integrated system can only function smoothly if data can be delivered to each of its components automatically and reliably.

  Data collected or processed by an application is always formatted to best fit the application's operation. Many existing applications were built without system integration in mind when integration was not possible. Therefore, a major task in application integration is to make data mutually understandable between the components of an integrated system. Making data understandable means converting its type and format to be accessible to a receiving application.

- **have each component act and react at the right time, in the right manner, automatically and reliably.**

  In an integrated shop floor control system, if a machine is down, the material handling system should not deliver parts to the machine as if it were in normal operation. The broken machine should be reported to the scheduling and maintenance systems and appropriate actions should be taken to correct the problem.

- **have a means to administer the whole system from a central control point.**

  Managing and controlling an integrated system would be a headache if a user-friendly and flexible system management facility were not available to assist the system manager in his/her system administration tasks. Imagine the effort required to configure, start up or shut down a distributed system if one has to go to each node in the network to perform these tasks.

In brief:

A truly integrated software system establishes communication between functional components, establishes control procedures for information exchange and activity coordination, automates control of various operations, and enhances human productivity with a coordinated presentation of the acquired or processed data.

In the next section we will examine in detail the functional and operational requirements for such an integrated system.

3. Application Integration Requirements

3.1 Functional Requirements

A truly integrated system must be capable of:

- Sending messages synchronously and asynchronously between components

- Providing mailbox services to participating applications so messages can be sent even if the receiver is not up and running

- Translating, manipulating and reformatting data to make a message readable by the receiving application
- Allowing an application to read a message in a destructive or a non-destructive mode
- Providing guaranteed message delivery
- Maintaining message sequence while providing a prioritization scheme for messages' access and/or transfer
- Sending files per request, per event or by time schedule
- Notifying receivers of arrivals of messages or files
- Monitoring and controlling processes in order to coordinate activities
- Restarting the system in an organized manner, e.g., without loss of in-transit data, in the event of a power failure
- Reporting data transfer system failures in an organized and useful fashion
- Taking corrective actions for non-critical errors so the system can keep functioning

In addition to the above basic requirements, there are many desirable features such as:

- The ability to move component programs around in the network without reprogramming
- The ability to recover from communication or control failures with minimal or no human intervention
- The ability to audit the integrated system activities to provide system integrity control
- The ability to diagnose problems in the integrated system
- The ability to simulate integrated system behavior for prototyping and testing
- The ability to provide a central point of control without creating a single point of failure

3.2 Operational Requirements

An integrated system having all the required features may not be a viable solution if certain critical operational requirements are not met. Some basic operational requirements are:

- Performance - An integrated system is not useful if its performance hinders the business operation.
- Reliability - An integrated system is not usable if it is not reliable. The requirements of a business operation define the reliability of an integrated system. For one operation reliability means recoverability but for another it may mean fault tolerance.

- Flexibility - A good integrated system should be flexible enough to provide dynamic reconfiguration and to support dynamic data structures. It should also be flexible enough to allow partial start up or shut down of the system.

- Security - A good integrated system should provide a way for the user to control access to all components in a manner acceptable to their business operation.

In conclusion, a truly integrated system must be built on an infrastructure that can provide:

- message and file transfer

- data manipulation and translation

- local and remote program control

- configuration management

Let us examine inter-process communication and remote-procedure-call and see how much these facilities can help in the implementation of a truly integrated software system.

4. Inter-process Communication

4.1 What is IPC?

Inter-process communication (IPC) is a means for processes to send messages to each other. IPC usually comes in the form of operating system or network service function calls. IPC facilities can be classified into two categories:

- local IPC

- network IPC

Let us examine each of these categories in detail and see how well they serve the application integration task.

- **Local IPC**

  The term "local IPC" refers to facilities available within one and only one computer system, not across a network. These facilities are usually operating system dependent. They are in most cases operating system features or sub-systems of the OS. The following are some examples of local IPC mechanisms:

  - **Signals** - Signals are simple IPC mechanisms. Signals are operating system calls that allow a process to be interrupted if it has set up a trap to intercept the signals. Most signals are provided for specific purposes and they carry with them specific meanings. However, some operating systems do provide general purpose signals for users to use according to their programming needs. Since signals do not contain much information in them, their usefulness in interprocess communication is limited. The communicating processes have to agree on the interpretation of a general-purpose signal. Signals are a very fast communication facility.

  - **Semaphores** - Semaphores, like signals, are simple IPC facilities. Semaphores are flags provided by an operating system for the communication and synchronization between processes. For some operating systems, semaphores are implemented as binary flags; for others they are numbers. Semaphores implemented as numbers are more flexible and can be used in several ways. However, like signals, the information that can be carried by semaphores is limited. Therefore, semaphores are of limited use for inter-process communication.

  - **Message Queue** - Message queues are a powerful IPC facility. They allow processes to send messages to each other using the queue as a mailbox. Message queues are mostly implemented using system memory; therefore, messaging performance through message queues is usually high. Some message queue facilities provide random access to messages. This gives some control over sequencing, concurrency access and synchronization, but at the same time reduces the flexibility of inter-process communication and impacts performance. Without random access to messages it is hard to implement a prioritization scheme.

  - **Shared Memory** - Shared memory is allocated blocks of memory accessible by multiple processes. Shared memory allows random access to data and usually provides better performance than message queues. However, without concurrent access control and synchronization (by way of semaphores, for example), shared memory is very unreliable since multiple processes can update the same memory location unpredictably. This makes shared memory management too difficult for inter-application communication.

  - **Pipes and Named Pipes** - Pipes and named pipes are other IPC facilities. They behave somewhat like message queues but with less flexibility.

- **Network IPCs**

  Network IPCs are facilities that allow processes to communicate across a network. A network IPC system is usually a set of network interface calls which allows a process to

  - set up a connection with a remote process

  - send a message to a remote process

  - receive a message from a remote process

  Examples of network IPCs are:

  - BSD 4.2 IPC (a.k.a Berkeley sockets)

  - HP NS NetIPC

  - IBM LU 6.2

4.2 IPC and Integration Requirements

Local IPCs are very useful for the implementation of modular applications. They can also be used for the integration of different local applications. However, their usefulness is limited by their single-system environment characteristics and by their dependence on the operating system. If one wants to split two applications from one computer to two computers, one must rewrite the interfaces.

Network IPCs are necessary for the integration of distributed applications. However, they only solve a small number of problems encountered in this area. Network IPCs provide the basic functionality for transferring messages across a network but there is still a lot of work to be done to turn a network IPC system into a reliable message handling system.

4.3 IPC Shortcomings

Shortcomings of local IPCs:

- The communication mechanisms are embedded in the applications' code. This leads to inflexible functionality and configurations because changing network addresses requires changes to code.

- The communication mechanism is operating system-dependent. The portability of the application to another computer architecture is limited or impossible. For the case of an integrated system, a component cannot easily be moved from one computer platform to another.

- The communication is available only between local processes. This is unworkable in the implementation of a distributed system with heterogeneous machines.

- There must be a predefined convention for local processes to communicate (communicating processes using message queues must somehow know and use the same queue ID; communicating processes using shared memory must somehow know how to attach themselves to the designated shared memory areas.).

- There is minimal or no error recovery provision for users of the local IPCs. Any failure recovery measure if needed, must be implemented in the communicating processes.

- There is very little or no access control and synchronization on the exchanged data (e.g., all processes sharing a shared memory segment can update it unpredictably).

- There is no provision for invoking a receiving process. If a message is in a message queue and the receiving process is not running, the message will have to wait. If the receiving process is running but not awaiting a message, there is no automatic way to notify the receiver of the arrival of the message.

- There is no data conversion mechanism for language-to-language data transfer.

Shortcomings of network IPCs:

- The IPC calls must be embedded in the application code. This results in network-service dependencies limiting portability and requiring complicated expertise.

- The management of connections must be handled by the communicating applications. Without connection management, resources will be wasted and if a connection is built for each act of communication, performance will be poor.

- There is little or no provision for error recovery. If reliability is desired, a recovery scheme must be implemented on top of IPC.

- There is no provision for asynchronous messaging; the receiver must be up and running or messages will be lost.

- There is no provision for data translation. Data sent between heterogeneous computers must be converted by the interface programs.

- There is no provision for manipulation and reformatting of data. Even if all other problems are solved, the receiving applications still cannot understand the incoming data.

- Some network IPCs (e.g., LU 6.2) provide for invoking a receiving process. This is not an invocation of a receiver, the communicating applications have to invoke the receiver before sending its messages. There is also no message arrival notification facility.

- Network IPCs do not provide a mailbox for staging messages. If this is desired, then the communicating processes must be implemented with message storage and management in their code.

- Network IPC mechanisms between heterogeneous computers are not uniform. This makes it difficult to have consistent application program interfaces and robust error handling and recovery mechanisms.

In conclusion IPCs, especially network IPCs, are necessary for application integration but there are still many requirements to be met in order to achieve true integration. The lack of functionality critical to integrating software applications is surprising. For example, the generic message queue mechanism yields an intolerant application environment. A dead process cannot simply restart and reconnect to the rest of the applications in an integrated system. Most often, all processes will have to be killed and then restarted. This is also true for generic socket schemes across machines.

5. Remote Procedure Calls

5.1 What is RPC?

Remote Procedure Call (RPC) provides execution of a procedure or a subroutine by a foreign host with results returned to the local calling process. The main feature of RPC is that the subroutine invocation is the same whether the routine's code is executed locally or remotely. Thus, code does not have to be altered to take advantage of distributed processing. In fact, RPC makes a local routine looks like a remote one. The main contribution of RPCs is the use of foreign host's resources, predominantly computing power, shared memory and/or peripherals. The simplest use of RPC is to provide intrinsic access to distributed resources which are directly callable by an application (e.g. printers, plotters, tape drives for backup tasks, math processors for complex and time-consuming calculations etc.). A more efficient use of RPC at the application level might be to partition the application so that the software modules are co-located with the resources that they use. For example, an application which needs to extract data from a database could be partitioned so that the modules which access the database could reside on the database machine. One could also build a distributed database machine with RPC.

5.2 RPC and Integration Requirements

RPCs can be used as a medium for passing messages across machines. However, they are usually not designed for this purpose. The design goal for RPCs was not to be a tool for building a message handling system but rather a tool for building new distributed applications where a calling process needs not know if execution is local or remote or if remote, where it happened. RPCs do provide data translation capability so that data sent between computers of different architectures are in the representation of the receiving host. This does not mean that data will be in the format readable by a receiving process. To make data accessible to a receiving process, RPCs would have to have data conversion and reformatting capabilities. None of the existing RPCs possesses data reformatting capability. However, some RPCs do provide data conversion capability through a method of using an interface language. The data conversions offered are between types of a same programming language; no cross language conversion is provided. RPC requires that data types have to be predefined prior to the implementation of the linkage between two processes. Any time a change is to be made to any data type, new source code must be created and the communicating processes must be re-compiled.

5.3 RPC Shortcomings

The following are major shortcomings of RPC in regards to application integration needs:

- RPC is primarily a remote execution service, not a data transfer service. Therefore, it is very awkward to use RPC as a message handling system. The integration of existing applications needs features of a message handling system rather than those of RPC.

- RPC operations are synchronous (the caller suspends pending the completion of the remote execution). One process cannot send a message to another process without having to wait for a reply.

- RPC execution is typically a one-to-one operation (client/server relationship). One process cannot broadcast a message to other processes using RPC without making multiple calls.

- RPC requires arbitrary request/reply semantics (the client and the server must agree in advance on what the requests and replies will be). This means a linkage between two processes must be defined at system implementation time, not at system configuration time or at run-time.

- During execution, a remote procedure cannot tell when it will be invoked again;

therefore, communications are always initiated at the beginning of its execution and terminated at the end. The initiation and termination at every invocation make it very costly (performance-wise) for a remote procedure to set up a connection with its caller. That's why most RPC are connectionless using unreliable datagram protocol (UDP). Because a connectionless communication is not reliable good RPC systems must build another protocol on top of what UDP provides to ensure reliability. This adds overhead that detracts from performance.

- RPC does not support data structure reformatting.
- RPC does not provide a central control point for a distributed application.
- RPC does not provide configuration management.
- Both sender and receiver (client and server) must be running for data to be transferred. There is no mailbox facility provided with RPCs.
- Due to the synchronous nature of RPC, it is hard to achieve parallel tasking with RPC.
- RPCs do not support program control. Therefore, you have to do it yourself.
- RPC does not provide location transparency at source code level (a program using RPC must have the address of the RPC routines).
- RPC does not support file transfer.

In conclusion, like IPCs, RPCs are not sufficient for application integration. As a matter of fact, worse than IPC which is a necessary infrastructure for the integration task, RPC provides almost no useful functionality to integrate existing software applications. Instead, RPC is more appropriate for building new distributed applications.

Let us now discuss a software tool designed for application integration.

6. A New Application Integration Tool

For software system developers the opportunity to build an integrated system from scratch is rare. In most cases, customers are tied to their existing applications. Their business operations become dependent on those stand-alone systems, often called point solutions, to a degree that it would be impossible or very costly to replace them. There is only one acceptable solution and that is to provide linkages between these applications. Building hard-coded links would result in an inflexible system. Building a flexible integrated system without a software tool would take a long time and would be very costly. An obvious conclusion to this problem would be a good software integration tool.

A good software tool for application integration can shield the system integrators from the vagaries of heterogeneous computer architectures, operating systems, languages, network services and application specifics.

HP has recently introduced the HP Software Integration Sockets (HP Sockets) product in order to provide this necessary integration tool to HP customers.

HP Sockets is a software tool designed specifically for the integration of existing and new applications in a network of heterogeneous computers. The main goal of HP Sockets is to integrate applications with minimal or no changes to them. HP Sockets provides functionality superior to that of a combination of a message and file transfer system, a data manipulation and translation system, and a local and remote program control system. HP Sockets also provides system management and logging of data and errors to ensure the reliability, auditability and recoverability of an integrated system.

Communication functionality provided includes connection and session management, time-out handling, message-spooling, message-sequencing, message- tagging, guaranteed delivery, incoming message and file notification, synchronous/asynchronous messaging, destructive/non-destructive message reads, mailbox and message access by tags or range of tags. Communication is provided through a table-based configuration to maintain flexibility and location transparency.

The data manipulation and translation capabilities of HP Sockets preserve data models of existing applications. Data types and structures can be different at each end of a messaging path. Thus a participating application in an integrated system can be replaced without affecting the rest of the applications and the replacement can be done without modification of application code. Any change in data elements, data types or format of a message only requires a reconfiguration of the system.

The program control functionality allows automatic start up or shut down of participating applications so that a system can function harmoniously as an integrated whole with minimal human intervention.

HP Sockets is the first tool designed and created solely for the purpose of integrating applications. It addresses all of the critical needs of a truly integrated system, as listed in section 3 above. With HP Sockets, system integrators will be able to:

- reduce the implementation time
- eliminate re-coding of the linkages should the
    - system requirements change
    - network services change
    - hardware configuration change
- reduce support and maintenance cost
- provide a simple and reliable method to manage the integrated system

To the end-users, HP Sockets will bring unparalleled flexibility in system configuration, will allow for incremental integration of existing islands of automation with less cost and will preserve existing investments.

7. Summary

In sections 4 and 5 we looked at IPC and RPC mechanisms and evaluated their shortcomings in regards to application integration needs. We summarize here major requirements that are not met by IPCs and RPCs:

- mailbox services
- message prioritization
- notification of arrival of data
- guaranteed message delivery
- location transparency
- file transfer
- data manipulation and reformatting
- local and remote program control
- system management capability
- system failure reporting and recovery

Considering this serious lack of functionality, we can conclude that neither IPCs nor RPCs are sufficient for rapid development of flexible and truly integrated systems, and HP Sockets is the only tool available today that meets the critical requirements of application integration.

A Step Beyond CIM: A Group Technology Approach to System Development

Theory and Experience

Feyzi Fatehi

Industrial Applications Center
Hewlett-Packard Company
1266 Kifer Road
Sunnyvale, California 94086

Abstract

Traditionally, industrial automation systems have been built from the bottom up resulting in the creation of islands of automation and information. Consequently, creation of these islands has resulted in a significant amount of redundant functionality, as well as redundant design, coding, and testing effort.

This paper proposes the use of *Group Technology* principles to significantly enhance the system development productivity and the resulting system quality, efficiency and flexibility.

The successful results of applying this methodology at the micro-level to the development of HP Real-Time Database are also discussed.

INTRODUCTION [1]

Group Technology (GT) has become increasingly important in today's manufacturing environment. One reason is that the full implementation of group technology can eliminate redundancy across most areas of manufacturing such as design, process planning, scheduling, factory layout, assembly, machine utilization, procurement, and quality assurance.

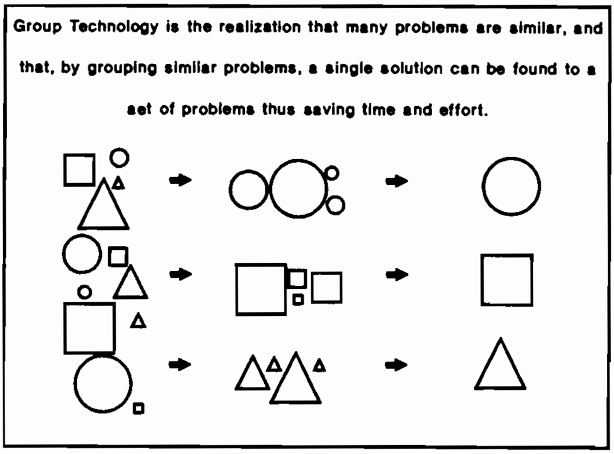
Group technology has been defined as "a manufacturing philosophy that identifies and exploits the underlying similarity of parts and manufacturing processes[2]." As conceptually depicted in Figure 1, GT groups parts into families by means of coding their characteristics such as geometrical aspects, manufacturing processes, and required tooling, equipment, and manpower.

Initially proposed by Mitrofanov[3] in 1959, GT has been widely developed and utilized in the manufacturing industry. Among its many benefits, it has resulted up to fifty percent increase in engineering productivity.

When studied carefully, one realizes that group technology is not really a technology, it is a metholology. It is logical to categorize and associate things based on attributes and features that they have in common. In this sense, the principle of group technology can be widely used and applied to areas outside the realm of manufacturing.

Even in manufacturing, an extended definition of group technology is emerging that encompasses a larger view of its applicability than to the actual shop floor and the tooling machines.[4]

In this paper I propose the extension of group technology principles to system development in general and to industrial automation applications in specific. The term Computer Integrated Manufacturing (CIM) is used here as a frame of reference, while acknowledging that the concept of CIM can easily be applied to other *nonmanufacturing* industrial automation applications simply by redefining the word "manufacturing" to apply to the integration of the informational needs of the intended industry.



Group Technology is the realization that many problems are similar, and that, by grouping similar problems, a single solution can be found to a set of problems thus saving time and effort.

Figure 1. *The Group technology concept*

ISLANDS OF AUTOMATION

Traditionally, industrial automation efforts have been focused on automation of individual activities resulting in the creation of islands of information and automation incorporating computerized technologies. The full benefits of the new computerized technologies are only realized when the full integration of these islands has been acheived. Lack of such integration, requires manual transfer of data from one automated island to another, creating inevitable opportunities for data redundancy. Data redundancy not only results in data inconsistencies, but also causes a significant overhead due to redundant effort for data collection, verification, synchronizing, and storage.

COMPUTER INTEGRATED MANUFACTURING (CIM)

From this prespective, CIM is the integration of the islands of automation. Integration can be defined as the availability of the information required by each activity on a timely basis, accurately, in the format required, and without asking.[2]

The challenges to CIM are not solely technical, but also cultural. The technical challenge is the complexity of the technologies involved, further complicated by the number of different vendors, the incompatibility among systems, as well as the lack of well-adapted standards for data storage, formating, and communications.

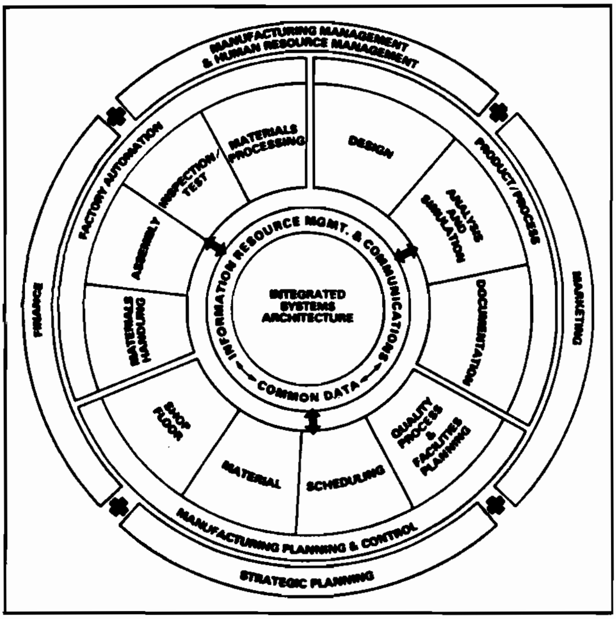


Figure 2. *The CIM Enterprise Wheel*

The cultural challenge comes from the fact that integration of the islands of automation requires organizations to view their operations as a whole and not as fragmented functions. This requires redefining the scope of the manufacturing task beyond manufacturing.[5] The emerging definition of manufacturing includes activities within many different functional units such as engineering design, manufacturing engineering, process planning, marketing, finance, operations, information systems, materials control, field service, distribution, quality, and production planning. Recently the term Integrated Manufacturing System (IMS) has been coined for those who prefere not to redefine *manufacturing* in CIM, but instead use a new term to refere to the plant-wide functions included in the expanded definition.

Developed by the Computer and Automated Systems Association of the Society of Manufacturing Engineers, the CIM Enterprise Wheel, shown in Figure 2, was designed to provide industry with a common vision of the scope and components of computer integrated manufacturing.[6] The model consists of five fundamental dimensions: business management, process and product design, factory automation, manufacturing planning and control, and common information resource management. Each dimension is populated by a family of products which has emerged naturally because of an affinity among what used to be stand-alone islands of automation.

STEPS BEYOND CIM

What steps are there to take beyond the full integration of all islands of automation incorporated in the CIM enterprise model? There are not many answers to this question, but the few that exist, almost unanimously focus on depth rather than breadth related issues.

The answers cover areas from just-in-time manufacturing (JIT) and total quality commitment (TQC), to topics such as eliminating wasted time, motion, and materials, shortening cycle responses, and creation of interdisciplinary produt and process teams.[5] Increasing the flexibility of the CIM enterprise has also been a major focus for system integrators, moving them away from the creation of rigid, monolithic systems.

These increasing needs for flexibility has been translated to the *Open System Architecture* concept. Implementation of this concept provides system integrators with "the freedom to build their systems incrementally, thereby taking advantage of modules with the best price-performance, which provide the required feature set for any of their particular applications[7]." An Open System Architecture compliant CIM model also increases the level of substitutability of CIM components in order to either enhance or modify the functionality of the integrated solution in a specific area. This way a *CIM Architecture* that has been developed for a large plant in industry A can be modified and reused in a small plant in industry B.

APPLICATION OF GROUP TECHNOLOGY

In taking a step beyond CIM, the untapped area of eliminating redundancies is an area with a great potential for success. Reducing and eventually eliminating redundancies has been a major area of concern for both the CIM venders and their clients. In almost all implamentations of CIM, a great amount of redundancy exists in overlaping functionality of member products, as well as in the internal design and development of each product. Unfortunately within the realm of CIM, group technology has only been utilized to eliminate redundancies on the shop floor and for the tooling machines.

Group technology can be used to simplify and specify new boundaries for a new generation of CIM products. These *CIM-compliant products*[8] will not only eliminate redundancy of design and development internally, but also eliminate functional overlap with other CIM-compliant products within the CIM enterprise. In this sense, group technology will facilitate a systematic definition of commonalities to guide the future enhancements and development efforts.

Nevertheless such improvements and efficiencies will require a genuine commitment to drive and mangage an orchestrated effort by industry leaders. Defining CIM-compliant products for industry architectures with no functionality overlap, requires industry architects with a collective understanding of an industry, its existing standards, as well as its emerging needs and requirements.

Eliminating the redundancy of effort from internal design and development of CIM products also requires software architects with a global view of the internal design to drive application of group technology to enhance the internal reusability level of design during the product development life-cycle.

Like implementing CIM, application of group technology can either be a highly focused effort which can take up to several years for an enterprise, or it can be an evolutionary process, phasing implementation into those areas where the highest returns are expected and then slowly expanding into the other areas of the system.

Application of group technology and implementing CIM also share the same issues in the areas of cost-justification and cultural resistance. Neither economic justification of CIM, nor application of group technology can be performed by traditional methods, which emphasize direct labor savings. In the case of applying GT, this is due to the fact that in the short-run it requires diverting labor from doing *productive* work to the task of identifying the commonalities, i.e., extra work. Promoters of GT applications may also face cultural resistance. Giving visibility to the long-term benefits, as well as other means of educating the decision makers and implementors, can help overcome both barriers.

THE CASE OF THE HP RTDB PROJECT [9]

Application of group technology helped the Hewlett-Packard Real-time Database with on-time delivery of a superior product. Developed at HP's Industrial Applications Center (IAC) in Sunnyvale, California, HP RTDB has been one of the most successful recent software development projects at HP.

Of course many factors contributed to the success of the project, including on-time delivery, exceeding performance goals, meeting resource consumption targets, zero defects reported after more than a year of utilization, and excellent product documentation.

But a major contributing factor was internal reuse. The project team believed that one of the best ways to increase both the productivity and the quality factors was to reduce redundancy. To achieve this goal, the concept of group technology was utilized during the internal specifications phase of the product life-cycle.

A group technology approach

In conventional software engineering practices, after the completion of the external specifications and a high-level design, the system is divided into sub-tasks or modules. These modules are then assigned to different members of the development team, who in turn decomposes each into more manageable tasks and defines the required interfaces between them. The success of the final system depends on the successful integration and interface between all identified tasks and sub-tasks. This methodology, which is commonly practiced, often results in a significant amount of redundant code in the final product.

In contrast to conventional programming that focuses on the relationship between the software engineer and his task, the HP RTDB Project focused on the relationship between sets of *common* modules/structures and their clients.

Three stages of the internal design

The internal specifications (IS) development phase of the HP RTDB Project was divided into three distinct stages:

Decomposition of the global system

First the global system architecture and behavior was designed, then functionally and structurally decomposed. In functional decomposition each high-level module, identified and defined during the external specifications phase, was decomposed into its sub-modules, along with the external specifications of each sub-module. This was done by each individual engineer for each of his/her assigned high-level modules.

In the structural decomposition all required data structures were identified and defined along with their required operations. Included in this category were structures such as system catalog, storage and indexing structures, and operations such as access methods, structure definition and deletion mechanisms.

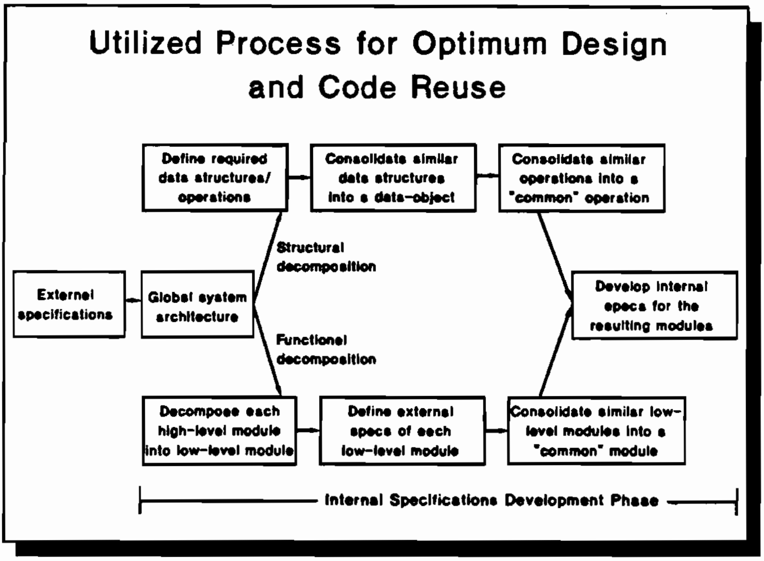

Utilized Process for Optimum Design and Code Reuse
External specifications
Global system architecture
Structural decomposition
Functional decomposition
Define required data structures/ operations
Consolidate similar data structures into a data-object
Consolidate similar operations into a "common" operation
Develop internal specs for the resulting modules
Decompose each high-level module into low-level module
Define external specs of each low-level module
Consolidate similar low-level modules into a "common" module
Internal Specifications Development Phase

Figure 3. *The three stages of the internal design: decomposition of the global system, consolidation of the reusable entities by applying the group technology concept, and the internal specifications.*

Identification of reusable entities

When the global decomposition was complete, the team focused on a consolidation effort, leading to sets of "common" modules, data objects and their corresponding operations, resulting in the elimination of redundant design and code.

This was a very demanding team effort, since it required everyone in the team to carefully study the external specifications of all the identified sub-modules, data structures, and their corresponding operations.

This process resulted in grouping all identified entities into two different categories: reusable and non-reusable. The non-reusable category consisted of structures and sub-modules that were unique and very specific, that were therefore none-reusable by others. Unlike the structures, most of the sub-modules in this category were bundled with their associated high-level modules and did not become a separate entity.

The reusable category included structures and sub-modules that could be used by more than one client, therefore eliminating the need for having more than one of them incorporated in the product. A combination of the reusable sub-modules and the operations associated with the reusable structures were referred to as *common modules*. The reusable data structures themselves were referred to as *data-objects*, borrowed from object-oriented analysis terminology.

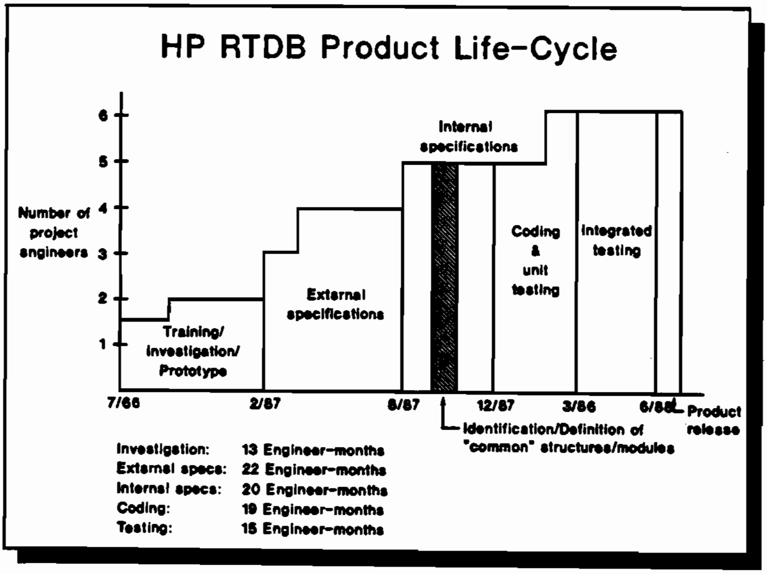


Figure 4. *Investment in identification, redefinition, and grouping common entities during design, pays off by reducing the effort required in the consequent stages of the development life-cycle.*

For example, the *table* structure became a data-object with all its associated operations. This data-object was reused by user-defined tables, hash indexes, and system catalog entities. System catalog entities also reused the hash index structure which itself reused table objects. This way a lot of redundant structures were eliminated along with many of their associated operations.

Most common modules were created by applying different amounts of modifications to an existing module, to accommodate for elimination of other similar but not exactly the same modules, using group technology concepts.

These modifications were mostly focused around the interfacing capabilities of these modules. In other words, input and output parameters of each module were adjusted with regard to the demands of all its higher level module clients.

Note that due to sensitivity of the performance goals of the final product, evaluating the performance consequences of any reusability decision was a decisive factor. The tradeoffs and possible diverse effects of reusing or modifying an existing module or structure to allow reusability were carefully evaluated before a decision was made.

Internal specifications

The last stage was developing internal specifications for all high-level and the resulting low-level modules. To develop the internal specifications we used structured English and/or pseudo-code where applicable.

Productivity gains

The investment in the process of identification and utilization of the common modules and structures directly resulted in:

- Reduction of the total code size by about 25 percent, resulting in less storage and memory requirements.

- Shortening the internal design/review phase by about 40 percent, since less internal specification was involved.

- Shortening the development period by about 40 percent, by eliminating duplicate coding effort.

- Shortening the period for code inspection and testing by about 40 percent, since less code was involved.

- Facilitating documentation, support, and maintenance since fewer modules are involved.

The above results, a direct consequence of applying GT concepts, increased the total productivity of HP RTDB project by a conservative estimate of 25 percent.

CONCLUSION

Application of group technology to system development results in significant elimination of both inter- and intra-product redundancies. This will lead to a significant improvement in break-even-time, higher productivity and quality, and reduction in cost of development and ownership.

ACKNOWLEDGEMENTS

I would like to thank members of HP's Industrial Applications Center whose suggestions, contributions and support made this work possible.

REFERENCES

[1] The first draft of this work was published in the proceedings of
IEEE Industrial Automation Conference
June 19-21, 1990 Toronto, Canada

[2] Glenn A. Graham, CMfgE.
Automation Encyclopedia, fourth edition
Society of Manufacturing Engineers, 1988

[3] Mitrofanof, S.P. Scientific Principles of Group Technology.
English translation by National Lending Library for Sc. & Tech. 1966

[4] Richard L. Diesslin
"Group Technology" article in Manufacturing High Technology Handbook.
Marcel Dekker, Inc. New York and Basel. 1987

[5] John Young, President and CEO, Hewlett-Packard Company
CIM and Beyond: Global Challenges for US Manufacturing
Spring 1990

[6] Daniel S. Appleton
Introducing the New CIM Enterprise Wheel
Copyright CASA/SME, November 1985

[7] Feyzi Fatehi
Hewlett-Packard Real-Time Database
Proceedings of Interex Conference
September 11-14, 1989 - San Francisco

[8] Internationa Business Machines Corporation
CIM in IBM
U.S. Marketing and Services, October 1989

[9] Feyzi Fatehi
R & D Network, Quarterly publication of Hewlett-Packard
Research and Development. Nov-Dec 1989 / Jan. 1990
Excerpts from "Group Technology + Design for reusability = on-time
quality software" article.

Much More Than Memory - the HP 1000 RAM Disk

Rebecca L. Carroll
Hewlett-Packard Co.
11000 Wolfe Road
Cupertino, CA 95014

INTRODUCTION

At the 5.2 release of the RTE-A operating system, disks are now allowed on a memory-based system. One may wonder, how can this be? These newly supported disks are actually not disks at all but memory disguised as disk space – RAM disks. The RAM disk is an exciting and useful new addition to the HP 1000. It has many applications in both memory-based and disk-based systems.

WHAT IS A RAM DISK?

To put it simply, a RAM disk is a driver which uses memory to emulate a disk drive. No special hardware is required, just available memory. Many operating systems offer RAM disks, but on the HP 1000, the RAM disk is much more than just memory; it is a powerful yet very simple tool.

Why use it?

RAM disks offer exciting possibilities for memory-based RTE-A systems. It provides a disk to a disk-less system, allowing that system to function much as a disk-based system might function. Now it is possible to adapt various disk-based applications to memory-based environments. For example, NS-ARPA/1000 requires a disk in order to bring up and execute the software. As of 5.2, a user can define a RAM disk during the BUILD of a memory-based system and, additionally, pre-initialize it with files before bringing up the system (on any other system, the RAM disk comes up empty; the RAM disk on the HP 1000 can come up containing data!). As a result, NS-ARPA/1000 can be booted on a memory-based system with all the files it needs for start-up at its fingertips!

There are also many reasons a disk-based user may want to take advantage of a RAM disk. The foremost reason is speed. Accesses from memory are certainly much faster than from disk (response is in microseconds versus hundreds of milliseconds), so it is wise to use a RAM disk for temporary storage of data or files. This may also be desirable for collections of data or for files accessed on a regular basis. RTE-A shows performance improvements when programs such as CI, LINK, LI, or DL are placed on a RAM disk; creating prototype-ID segments for these programs then provides even greater improvements to system performance.

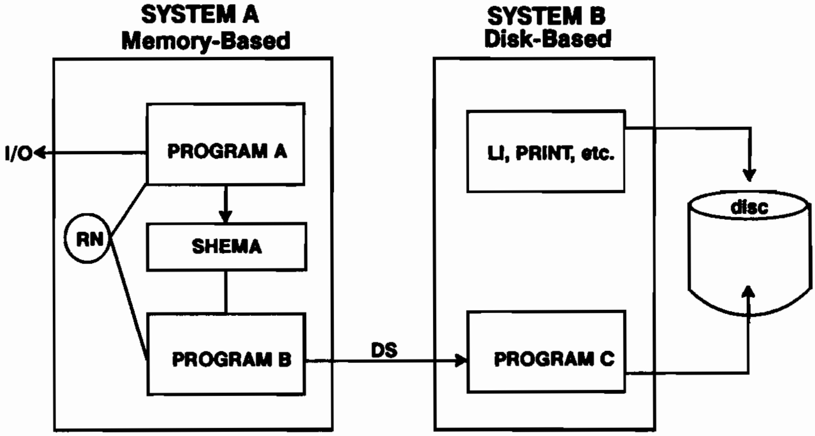
BENEFITS FOR MEMORY-BASED SYSTEMS

Simulation of a disk-based system

A memory-based system can be made to look like a disk-based system by including a RAM disk in the system BUILD. As long as both disk-handling and memory-handling routines have been included in the system generation, files can be accessed on the RAM disk just as they would be accessed on a real disk. The most outstanding feature of the HP 1000 RAM disk is that it gives a user the ability to create directory structures with files under them using the BUILD utility and then access those files on boot-up. Regular EXEC and FMP calls can then be used to perform I/O to the new "disk", until the time the RAM disk memory is released to the system.

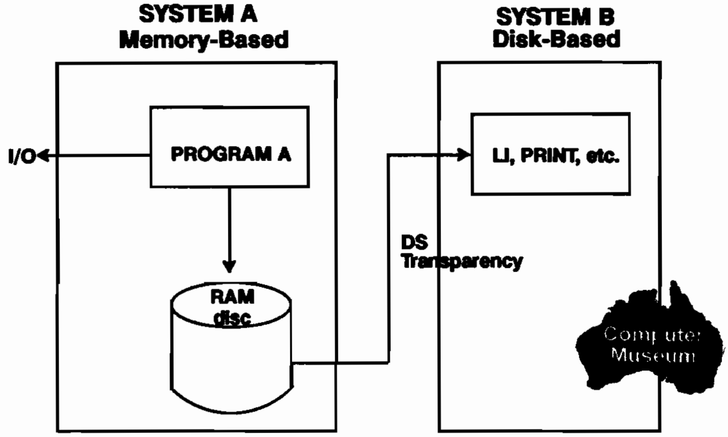
A simple example illustrates the benefits of the RAM disk. Assume a system (System B) needs files from another system (System A) in order to execute some command. System A is memory-based and has a program (program A) doing I/O; System B needs the output from this program. Program A, which isn't written to do any kind of DS communication, needs to put its output in SHEMA so another program (program B, which is able to handle DS) can transmit the data to System B. A resource number is also required for these two programs in order to lock SHEMA when necessary. A third program (program C) is needed on System B in order to receive the data through DS program-to-program communication and give it to the command which requested the information in the first place. To summarize, we now have two more programs in addition to our initial program (which provided the data) in order to use DS to transmit our information back to the request on System B. Whew! Quite a bit of work to get data from point A to point B (see figure 1).

FIGURE 1: Without a RAM disk



Now assume the same set-up but with a RAM disk on System A. The program doing I/O on System A can put its output on the RAM disk instead of in SHEMA. Additional programs are no longer necessary since a normal FMP call from System B's command can use DS Transparency to access the output directly from the RAM disk (see figure 2). This improvement has increased our productivity by an order of magnitude over the configuration without the RAM disk.

FIGURE 2: With a RAM disk



Application support

The ability to have a disk in a diskless system provides the user with a much wider range of software from which to choose. Subsystems which could never operate without a disk can now be run under a memory-based system. The most significant of these subsystems is NS-ARPA/1000. This software is very useful for certain applications, but it cannot be run without a disk.

There are two ways NS-ARPA/1000 utilizes the RAM disk. First, it is important to have all servers available on a disk. They must be cloned as they are used, which means they must be present somewhere for the system to access. For this reason, many NS-ARPA/1000 services will not work in a memory-based system without a RAM disk. Secondly, networking software uses an initiation file (NSINIT) to start up . During the boot process, this file is entered if it is present in order to find host names and destinations. If this file is not found, the information must be entered interactively. Since the information is cumbersome to type in, it is advantageous to provide it to the system during boot-up. In order to facilitate the start-up of a memory-based system, this file can be loaded on a RAM disk during BUILD's initialization of the disk; our system boot is now much more efficient!

Other ideas

There are other ways the RAM disk can be utilized in memory-based systems. In addition to the NSINIT file mentioned above, it may be helpful to have the NS message files available to decode those cryptic error messages. NS and DS both benefit from having a nodenames file on RAM disk for easier manipulation of nodes. The I/O program uses a file called devices on the /SYSTEM directory which assists the user in identification of devices. Placing these and similar files on a RAM disk makes life much easier in a memory-based environment.

There is a trade-off, however. The inclusion of a RAM disk decreases the amount of memory available on the system by the size of the RAM disk. This must be taken into consideration when planning a memory-based system. The benefits of a reasonably-sized RAM disk, however, outweigh any disadvantages.

BENEFITS FOR DISK-BASED SYSTEMS

Performance improvements

A disk-based system will see performance improvements in two main areas, program dispatching and program execution. In the area of dispatching, the inclusion of type-6 files on the RAM disk allows the system to access the program directly from memory instead of from disk. This minimizes the amount of swapping that occurs in the system. Providing prototype-ID segments for the programs on RAM disk also keeps the system from going to disk for the ID segment; the application is quickly dispatched, and execution begins immediately.

The following is a sample transfer file to configure a RAM disk. In addition to the programs mentioned below, user-written applications that are frequently used could also be placed on the RAM disk for better performance. Note that this file could only be run once per boot–up because the RAM disk driver, IDR37, will not allow the disk to be reconfigured once it is already set up. The following commands can be used to configure any RAM disk LU by changing the underlined numbers to the desired LU number; the commands used below will be described in a later section.

```
echo `configure RAM disk on LU 2`
cn 2 25b 2000
*
if mc,2
   then
   echo `initialize LU 2`
   in 2,,ok
*
   echo `create directories on LU 2`
   crdir /ram 2
*
   echo `copy files to directory /ram`
   co /programs/ftn7x.run /ram/
   co /programs/dl.run /ram/
   co /programs/ci.run /ram/
```

```
        co /programs/cix.run /ram/
        co /programs/li.run /ram/
        co /programs/link.run /ram/
        co /programs/macro.run /ram/
    fi
    echo `RP RAM disk programs (with proto-ID segments)`
        rp /ram/ftn7x.run,,d
        rp /ram /dl.run,,d
        rp /ram/ci.run,,d
        rp /ram/cix.run,,d
        rp /ram/li.run,,d
        rp /ram/link.run,,d
        rp /ram/macro.run,,d
```

As mentioned earlier, performance is also improved in the area of program execution. For example, the compilers for Fortran-77 and MACRO/1000 put their working files into the /SCRATCH directory. When /SCRATCH is on a RAM disk, the compilers execute more efficiently since they can immediately access their working files from memory rather than from disk. Any other programs using /SCRATCH in a similar way would also benefit. Note that Pascal does not work this way; it puts its scratch file into the current working directory instead of into /SCRATCH.

Trade-offs

In addition to the previously mentioned benefits, there is one driver only for the RAM disk (and it is small), so relatively little space is needed in the system itself. There are, however, a few things the user should keep in mind. Since the RAM disk is part of memory, everything on the disk will be lost if a powerfail occurs and battery backup is not on the system. Therefore, the user should carefully choose which files are stored on a RAM disk. Another consideration is that the driver, although small, uses a moveword instruction in its operation, resulting in the occasional hold-off of interrupts. This is of concern to a real-time system developer. Finally, a balance between memory space and RAM disk space must be found. An active system could actually see more swapping than normal if a large RAM disk were configured. Probably the best way to determine the optimal size for the RAM disk is trial and error. Ways to check for the amount of memory that is generally free might be to run WH,PA several times throughout the day or to write a program to sample memory on a regular basis. A way around the balance issue is simply to buy more memory. The price of memory combined with the increased performance of the RAM disk can make this option a worthwhile one.

HINTS AND TIPS

The utilities EDIT and WH have both been modified to take advantage of RAM disks. Their enhancements are discussed below.

EDIT

EDIT has been modified in the way it works with scratch files. Remember that any files on a RAM disk will be lost if there is a powerfail and battery backup is not on the system. This means if /SCRATCH is on the RAM disk and an edit was being done on a file, those changes will now be lost. In order to continue offering those changes to the user with a RAM disk, EDIT will now detect the LU of the select code where the directory /SCRATCH has been found. If it is found on the RAM disk (the select code is zero), EDIT attempts to use the directory /SCRATCHEDIT instead for its work files. To be sure an edit can be recovered in the event of a power down, it is recommended to create the directory /SCRATCHEDIT on a real disk.

WH

Executing WH,PA shows the layout of memory in the system. If any RAM disks are present, WH now displays them in the layout; they are listed last since they are at the end of memory. A question might be, how does WH know the difference between bad memory and RAM disk memory?

WH determines how memory is allocated by looking at the memory descriptors associated with the memory blocks. For a RAM disk, word 4 of the dynamic memory descriptor, instead of pointing to the next free block, contains the LU number of the RAM disk. This LU number is used to index into the LU table and find the DVT associated with the disk. The address in driver parameter 3 is the address of the memory descriptor for this LU. If this address matches the address of the memory descriptor in question, then WH has found the RAM disk and will include it in the report. If they don't match, this memory descriptor is describing bad memory (such as results when a parity error occurs).

HOW DO I PUT IT IN MY SYSTEM?

Very easily!

There are two simple steps in the installation of a RAM disk. The system answer file needs to be modified to define the RAM disk, and memory needs to be dedicated to the disk. The only requirements are the 5.2 release or later of RTE-A and available memory.

There is no device driver for the RAM disk so the first step will be that much shorter. The interface driver is IDR37. The following lines must be added to the driver relocation section of the answer file:

```
    RE, /RTE-A/IDR37.REL
    END
```

Next a DVT and IFT must be specified for the RAM disk with the desired LU number as follows:

```
IFT, IDR37
DVT,,,DX:8,DT:33B,LU:xx
```

where xx = the LU number for the disk

Note that multiple RAM disks on one system are possible; however, a DVT is required for each RAM disk defined.

Next, the amount of physical memory devoted to the RAM disk must be determined. WH, PA can be run to see the total number of pages that are installed. The number of tracks available on the disk can be calculated by dividing the number of pages for the disk by the number of pages/track desired (default is 4). Note that if the tracks are long, a lot of space will be wasted for directories (which use only one page/track). However, a disk with a lot of short programs would benefit from longer tracks. The user must weigh these trade-offs when determining the number of pages/track to use.

Configuration is accomplished with the CN 25 command and can be done either interactively or in a welcome file. The command is used as follows:

```
CN lu 25B xx [yy]
```

where
lu = LU of RAM disk
xx = number of pages of RAM disk memory
yy = number of pages/track; must be no less than 1 and no greater than 8

NOTE: The number of pages of RAM disk memory (xx) must be evenly divisible by the number of pages/track (yy).

The RAM disk now looks like a real disk to the system, so it must be mounted and initialized like a real disk by either CI or FMGR. It can then be accessed by the user so directories and files can be created.

BUILDing it into a system

BUILD has been modified to allow the addition of RAM disks. It will not only define them but also initialize them. This includes loading files, building bit maps, and building directory information in addition to the actual definition of disk size and location. This feature of the RTE-A RAM disk is unique in the computer world; other RAM disks can be defined but not loaded ahead of time. BUILD also constructs the required memory descriptor for the RAM disk memory.

To inform BUILD that a RAM disk should be included in this memory–based system, the +D option is included in the runstring. This causes BUILD to create memory descriptors for any memory left over at the end of the build. After it has finished with the memory descriptor, the swap file information is cleared, and BUILD will prompt for files to be placed on the RAM disk. If files are listed, the user is now also prompted for the RAM disk global directory name. When a directory name is given, BUILD will once again prompt for file names, allowing for multiple global directories on the same RAM disk.

A note of warning: BUILD only knows about sixteen files at any one instance, so it is possible that the same file could be loaded as both the third file and as the seventeenth file without an error being flagged by BUILD.

Releasing the disk space

Releasing the disk space back to memory is accomplished with another CN command, CN 26; this must be done after the disk has been dismounted.

```
    CN lu 26B
```

Once the memory has been released, it is returned to the system's dynamic memory pool. If several RAM disks have been set up, it is best to release the last one first so the freed-up memory will be adjacent to the remainder of the system's free memory. Note that when the RAM disk memory is released, any files on the disk will be lost.

HOW DOES IT WORK?

And who are the drivers?

As was mentioned earlier, no device driver exists for the RAM disk. The interface driver, IDR37, manages all necessary manipulation of tables and memory. The following are the driver parameters used:

DVP1: starting page number for this RAM disk
DVP2: ending page number + 1 for this RAM disk
DVP3: pointer to the memory descriptor for this RAM disk
DVP4: not used
DVP5: not used
DVP6: number of tracks
DVP7: number of blocks/track
DVP8: not used

The driver supports EXEC 1, 2, and 13 calls. A look at the EXEC 13 returned status shows a driver type of 33B in STAT1 (a disk) and an interface type of 30B in STAT2 (the disk interface card). Note that the select code returned in STAT2 will be a zero for the RAM disk (this is the way to tell the difference between a real disk and a RAM disk!). The driver does not return a meaningful extended status. Details on EXEC calls can be found in the 5.2 *RTE-A Driver Reference Manual* (part no. 92077–90011).

A Look Inside

Setting up and using a RAM disk is very straightforward. However, what goes on behind the scenes is a bit more complex. An understanding of the driver itself may give a software developer insight into the way memory is actually being manipulated on the system. The following sections show the paths the driver takes during interaction with the RAM disk memory.

In summary, the driver allocates memory for the RAM disk from the free memory pool. The driver assumes the last memory descriptor in the list of descriptors holds most of the free memory, so it is wise to configure the RAM disk early in the life of the system. A memory descriptor is then created that marks this memory as bad (it actually looks the same as if a parity error had occurred). As described earlier, WH can detect the difference between down memory and RAM disk memory in order to include it in WH,PA reports. The driver then returns control to the user who puts the RAM disk to work.

IDR37

- **idr37**
  What does it do?:

  1. Get address of driver parameter area from $dvtp and sets up a pointer to the parameters we use.

  2. Jump to routine for specified entry directive. The only valid entry is initiate since we should not be receiving any interrupts (we always complete immediately).

- **initiate**
  What does it do?:

  1. On first entry to the driver, clear the sign bit in $dv20 (this was set by the generator as a First Time flag) and save map set 1.

  2. Get request code from $dv15; if a control request, jump to InitiateControl.

  3. If any other request type, check whether disk has been configured; if not, reject call (can't do anything until we're configured).

- **InitiateControl**
  What does it do?:

  1. Is this a CN 26 (deallocate) request? If not, jump to CN25.

  2. Get memory descriptor address (from dvp3) – does its starting page number match ours (in dvp1)? NO – reject request (something not set up correctly).

3. Load the A-register with the address of the memory descriptor and JSB to $MFRE to release the memory.

4. Check DVT address of swap LU – if it matches ours (from dvp1), clear it out.

5. Jump to ExitRelease.

- **CN25**
  What does it do?:

  1. Check to be sure disk has been configured – if not, reject call.

  2. Get number of pages/track and check that not fewer than zero or more than 8. If OK, check to be sure we are asking for an even number of tracks.

  3. Get starting address for the memory descriptors and run through the list until we find the last one (we assume the last memory descriptor holds most of the free memory).

  4. We have found the last memory descriptor – is it already in use? YES – reject call. Else, get the request size (from $dv16) and see if enough memory.

  5. If enough room, decrease block of memory by size of request, figure out where the block starts, set the block's first page address (in dvp1), get the request size, set the block's last page + 1 (in dvp2) and JSB to $GBLK to set up the memory descriptor. When it returns, save pointer to memory descriptor (in dvp3).

  6. Mark this memory as parity error memory and set status to locked. Now set the DVT link word to point to the next free memory descriptor.

- **read/write**
  What does it do?:

  1. Need to verify request length, track, and sector (from the DVT) against the values previously set up for the RAM disk:

     $dv17: length, from 0-32 KWords; can't go beyond last track
     $dv18: track, from 0-last track ((number of pages / number of pages per track) – 1)
     $dv19: sector, from 0 - (number of blocks per track * 2) –2.

  2. If length = 0, exit; else, if length not in words, convert it to words (100000b for 32 KWords).

  3. Get track address; multiply it by blocks/track (from dvp7) to get block offset, then convert this to a sector offset.

4. Get sector address and add in the sector offset determined in previous step.

5. Shift above value to get page offset, add the starting page (from dvp1), and save page offset.

6. Compute ending address in RAM – do we fit? (Check against value in dvp02.) If not, reject call.

7. Since we only support access to even-valued logical sectors, check to see if value in $dv19 is even or odd – if odd, reject call.

8. Check to see if we're bigger than the value in dvp7 – if so, reject call.

9. Can we map in all of the request at once? (We may not be able to map it in if request is bigger than 31 pages and begins somewhere other than the beginning of the RAM disk.) YES – skip to step 13.

10. NO – get negative of the page offset (this is the initial size to move to ensure second part of request starts on a page boundary).

11. Adjust request length in $dv17, adjust buffer address in $dv16 for next part of request, and do the move.

12. Set up next move with page offset as zero and bump the starting page.

13. Get count and address from DVT and do the move.

14. When finished with move, restore maps, clear driver communication bits, and exit.

- **ExitRelease**
  What does it do?:

1. Reset flag in $dv20 for first entry and set $dv17 to equal 8 * (number of blocks/track) (from dvp7).

2. (Now update the system memory lists, since we just either took or released memory.) Clear the variables that currently store the three largest blocks of memory. Collect all potentially available memory blocks – if none left, set the maximum size for the rest of the system and exit.

3. Now check all memory descriptors for size and see if it is now one of the three largest blocks. If so, update the appropriate variable. If not, keep going until we're out of memory descriptors.

4. Do final update to set the maximum size for the rest of the system and exit.

IN CONCLUSION...

The RAM disk is truly a very powerful and versatile tool for RTE-A users. It provides disk files to a memory-based system and significant performance improvements to a disk-based system. One of the RAM disk's most exciting benefits is the ability to be pre-initialized with a complete directory, including files, before it is brought up with the system. Even though there are a few trade–offs (nothing is ever free), the advantages of the HP 1000 RAM disk far outweigh any disadvantages. Enjoy using this new feature!

I would like to acknowledge Alan Tibbetts for his time and technical expertise.

Understanding the X Windowing System on the HP9000 / Series 800

Dan Dickerman

Hewlett Packard Company
19490 Homestead Road
Cupertino, California 95014

Abstract

With the recent popularity of windowing systems triggering a landslide of windowing products, consumers and "techies" alike have been thrown into a race to keep up with the new technology and new terminology that has emerged. This paper introduces the X11 windowing package on the HP9000, to help the user gain a better understanding of this unique type of software, and how its different pieces interact to present a unified windowing system. With a better understanding of X's underlying methods, it will prove easier to set up a working windowing environment, troubleshoot, and tune the system to provide the most efficient use of resources.

Life Before Windows

Until recently, the concept of how a computer should present itself had remained relatively unchanged since the 1970s. "Computers" have always been large mainframe machines that sit somewhere in the basement, and are connected to users through terminals, each with its own 80 column by 24 line display of ASCII characters. One of the problems with these terminals is that all information displayed on them had to be presented in exactly the same way: since basic terminals are only capable of displaying text, that's all the software has with which to work.

With advancements in electronic technology, graphical terminals started to come into common use, each with its own set of convoluted command sequences to produce system-dependent graphical output. These also were able to function in a nongraphical capacity, presenting text much like their predecessors, bringing with this emulation the choice of applications being entirely text-dependent or graphic and incompatible.

In the early 1980s, the desktop microcomputer appeared in the home and small business market. These tended to be single-user machines, unlike the high-end multi-user mainframes, with little to offer most in terms of capacity or speed. As they evolved, however, high-end micros began to offer the power to compete with the facility of the mainframes, and had many of their own innovations, while still sitting on (or at least near) the desktop.

Dubbed "workstations" (most probably due to the fact that they first appeared as workplace tools for the more technical), many of these high-end micros rejected the concept of the standard text interface in favor of a graphical user interface (GUI) that would allow more flexibility in the user's interaction with the machine. The "desktop metaphor", developed at Xerox PARC and popularized in the home market by Apple's MacIntosh, was a bridge that allowed people to associate concrete objects with common abstract computer concepts. They had windows, menus, icons, pushbuttons and a mouse to add variety and coherence to the interaction. To a multi-tasking workstation, this also meant that some of the "sheets of paper" lying on the desktop could represent separate, concurrently-running processes, each a window peering into a different task. High resolution displays allowed workstation users to do on one desktop terminal what would formally have taken either several text-only terminals, or a rather complex use of job control (from those systems that offered it). In addition to this extension of multi-tasking, windows provided methods of interaction entirely new to the common computing market.

Windowing Environments

Some brief clarification of terminology is in order here. In this desktop metaphor, on the desktop (screen) there appear several sheets of paper (among other objects), each loosely termed a window. In early systems, a window was merely an area on the screen that acted as a simple terminal itself, with the advantage that there could be several terminal-type windows open (visible) on th same display at the same time, overlapping if necessary. In time, systems developed wherein windows could be used to display output to interact with the user using a more graphical manner.

Along with the mouse, a supplement to the keyboard to allow a straightforward way for 2-D coordinate data to be input to the computer, the "point-and-click" graphical environment was adopted by many manufacturers, but there was still need for more development. Despite many external similarities, windowing implementations were still highly machine-specific, and thus lacked one of the emerging UNIX world's key features: code portability. Windowing applications that were written in common languages like C were still filled with such machine-specific libraries (at best), that they were still bound to their original platforms.

Soon it became evident that although presenting the same interface had advantages from the user's perspective, programming that interface presented its own set of challenges, with complex demands being made on system hardware and software techniques. GUI problems became even more evident when manufacturers began to intentionally modify the user interface, making small changes in appearance to avoid copyright infringements.

Enter X

Seeing the need for a better approach, in 1984 students of MIT's Project Athena began the development of the X windowing environment, a system- and network-independent windowing protocol that grew out of Stanford University's "W". Currently popular in its version 11 revision 3 release (X11 r3 -- the MIT consortium, however has already defined r4), X was designed to offer the capabilities of all workstations to the programmer of any of them. Using X protocols and a standard programming interface, Xlib, the same windowing code could be written that could run on different manufacturers' platforms with the same visual results. Also, applications running on one machine could have windows displayed on other machines (regardless of manufacturer).

To get around the system incompatibilities, X was defined at a sufficiently low level as to allow great flexibility, but at a high enough level to avoid the machine-specific differences. It assumes that all workstations have some sort of display, capable of drawing a particular set of graphic primatives, a keyboard for text and a mouse (for more graphical input). Each display unit runs an X server, which is responsible for interpreting these protocols and implementing them on that particular machine. Also, by introducing this level of abstraction, the X server is able to respond as easily to its own graphical requests as to those generated by another machine across a network.

This idea of machine transparency has also led to the production of stand-alone X servers that run X, responding to X network protocols, but that run none of the "core" applications locally. By responding only to the graphical requests of the X protocols, neatly dividing the application into computation and graphical presentation. Using a separate machine for the X server is actually rather efficient, since the X server, with all its related resources, is generally a large (4-8 Megabyte depending on configuration) piece of software in itself. By separating this functionality from the host CPU, it allows specialized hardware to handle what would otherwise steal CPU cycles from the host.

Notice here that this reverses the standard notion of a client server arrangement. In many other applications, the server is the large mainframe off in a room *somewhere,* with the client being the small local computer making requests to the server. In X, the requests are made to the local display server by the software application, usually running on the high-power, hidden away CPU.

What X Is

The X server provides all of the X resources and responds to all X protocols directed toward it, but does little unless under program control. In its plainest, the startup X screen displays a stippled background with an "X" cursor representing the mouse's current position. The Server is now responsible for tracking the movement of that cursor to keep it in line with the movement of the mouse. Everything beyond this must come to the server in terms of an X protocol request to the server from an application. The server should be thought of as a storehouse of resources: it keeps track of all information about applications running on its display (including font information, physical characteristics of the display hardware, and attributes of all windows including their relationship to one another), and allows those applications the responsibility of receiving, manipulating, and responding to that information.

The server treats everything on screen as either a window or graphics within a window. This may seem odd to many who have used X and think of elements within X to be made up of windows, buttons, menus, sliders, text, icons and others, but these are all higher-level interpretations of X's basic building blocks. Windows are basically rectangular regions on the screen that may be considered as independent units. All text or graphic information is displayed in some window, and drawn using graphic-oriented requests. All communication between these windows and the user or applications is through either requests to the server by a program, or though **events** generated by the server (often in response to something done by the user) to which the application must respond. The application is responsible for determining which events it will consider interesting, and then responding to the events in an appropriate manner.

Windows are organized in a typical tree-like manner, much like the directory structure of the UNIX system, with each newly-created window inheriting default attributes from its parent. The full-screen window is referred to as the "root" window, with all applications generating 1st-generation children of the root window (except with "re-parented" windows: see window managers, below). These child windows can also have their own child windows to further subdivide into independent display areas, buttons, menu's etc. Each window is labeled with whichever events it considers interesting, and all others are propagated through the parents. This means that if you were to type a key on the keyboard in a window that was acting as a button, that keystroke would be ignored by that button window, then its parent window would have the option of receiving that particular keystroke. If it's parent (and so on within that subwindow set) decided that it wasn't interested in that event, it would pass all the way up to the root window. At each step along the hierarchy, windows are given the opportunity to receive an event or pass it to its parent.

For example, consider a window on the display that contains a rectangular button. This scenario contains two windows: the large parent window (a direct child of the full-screen "root" window), and a smaller window representing a button, possibly with some text drawn in it to give it some meaning. The application that would be interested in the "pushing" of this button would tell the X server that, for this smaller window, it would be interested in "XButtonPress" or "XButtonRelease" events from the mouse. The X server would then relay all information about mouse buttons being pressed while in this window to that application, which is then responsible for doing something about each XButtonPress event associated with its windows.

Window Session Managers

There are also functions that are common throughout all applications running under a windowing environment; things that are more organizational or administrative, which are not part of the application's function, per se. Things like changing the stacking order of overlapping windows on the screen, generating menus of applications, creating icons for not-currently-active windows, allowing the user to move or resize the windows are all functions of the **window manager.** Without this piece of software, changing windows' positions, sizes and stacking order on the screen, as well as several other "niceties" that are commonly associated with the desktop model, would be considerably more difficult. Again, it is important to stress that the X server just keeps track of information, relaying it to the various applications running on its display, but doing little with the information on its own. Window managers take on these organizational tasks, then rely on the X server to communicate any relevant changes to interested applications (in the form of events), so that they may respond appropriately.

Window managers (although the term is not rigidly defined) usually allow the user to use the mouse to reposition and resize windows, and provide the user with menus to more easily select applications to run. Some add decorations such as title bars and frames with their own set of buttons and tools, and many allow the user to "shrink" the window down to a small **icon,** so as not to clutter up the screen, then to bring it back to full size again when it's needed. In creating a frame or title bar for a window, the window manager will create its own window, slightly larger than the application's window, and "re-parent" the application's window with it's own, surrounding the application's window with its own decorations. In the repositioning and resizing of windows, the window manager may expose or conceal regions of windows on the screen, but the actual redrawing of the information in these windows (or scaling to adjust for a new size) must be done by the applications themselves, in response to events generated by the X server.

Most window managers also keep track of the current "keyboard focus" and "colormap focus" (on color displays). Unlike the mouse, which has coordinates implicit in its input method, workstations usually only have one colormap and one keyboard for use on the whole screen, and so need a way to determine which window will receive the input from the keyboard or determine which colormap to use. The window manager is usually responsible for keeping track of which window will receive all keyboard-related events, and also of which application's colormap to use. The implied method is usually to give all focus to whichever window contains the mouse pointer, but window managers (or other applications) have the ability to change this to suit the needs of the user.

To give an example of what the window manager is doing, consider a common feature of a window manager: generating **icons**, usually small boxes on the screen to represent an application that may take up much more space. When the user tells the window manager to "iconify" an application,[1] the window manager program will tell the X server to "unmap" the window from the screen (stop displaying it without deallocating other resources), and to "map" a small window (the icon) in its place, which the window manager had prepared earlier, planning for just such an occasion. To complete these requests, the X server will take care of the redrawing of the root window that may have been exposed, as well as the rectangular borders and backgrounds of windows that are also exposed. It then will re-draw portions of windows that it had saved to its own temporary memory ("backing store"), if possible, and signal each relevant application that an area of its window is now exposed, and should re-draw that part. These **expose** events are also sent to the window manager application that started the whole sequence, so that it may know to start drawing inside the icon window. All applications that receive these expose events then redraw the newly-exposed portions of their windows.

The Programming Interface

As with all base-level protocols, a higher level of interface is necessary in order for programs to interact with them in any substantial way. X provides a structure wherein the base window system is accessed through the X Network Protocol, which can be accessed programmatically though the Xlib Programming interface. The Xlib interface is the standard way in which C programs can communicate with the X

[1] This is usually done though by using the mouse to "click" within a button-widget, which the X server sends to the window manager in the form of a "XButtonPress" event. Of course, the user sees this as "pressing a button". Depending on the window manager being used, this "iconify" request can also be made through a key-stroke sequence that is recognized by the window manager program (in the form of a "XKeyPress" event).

server. It provides routines that create, organize and manipulate windows and their associated attributes, as well as the buffering of X protocol requests for optimization. Using the Xlib interface, an application can access any of the X protocols in such a way as to make the application as specialized as possible. Note that since Xlib generates X Network Protocol requests, an application build with Xlib on any platform will be able to use any X server (at least any of the same revision) as its display, either locally on the same platform or across a network to a different manufacturer's platform.

The application has the choice of making *one-way* or *round-trip* requests to the X server. One-way requests are usually output primatives and other things that can be buffered into groups and sent as larger packages for efficiency. Xlib buffers one-way requests automatically, and allows for for users to explicitly flush the buffers when necessary. Round-trip requests are those that must wait for the server to respond before continuing. These include requests for resources and queries to parameters: asking the server for a display connection pointer or parameters about a particular window, for example.

Xlib provides standard basic graphic primatives, but still does not provide for the common objects of the window-oriented environment. For these, there are libraries built on top of Xlib which can provide a uniform set of objects usually termed "widgets" or "gadgets". Packages such as the X Toolkit and XtIntrinsics provide these basics for grouping objects into common classes. Higher-level packages such as OSF/Motif provide standard interface objects, building a groundwork of pre-defined widgets on top of the toolkit-level library to provide simpler ways of constructing a uniform collection of widgets across libraries of software.

The Difficulties with Xlib Programming

As we have seen, X in itself is a much lower-level interface than one might expect to find in a windowing system. It provides ways in which input can be taken from the keyboard or mouse, graphical primatives, organization of windows into independent entities, and communication between applications sharing a common display. Substantial as this is, due to X's desire for flexibility, Xlib does not provide an interface to the most common aspects of windowing systems: there is no such Xlib concept as "open a window for displaying text with a vertical scroll bar" (which would be like a normal, scrolling terminal window); "Create a pop-up menu with these selections"; even "Input a string from the keyboard" becomes a complex task for the Xlib programmer, involving reading XKeyPress events from the keyboard, associating them with the application's window, then decoding that keypress into the desired text.

All objects drawn in X are formulated in terms of graphical primatives, and all input to an application comes in the form of X events. For example, text is drawn to a window using a particular font resource, drawing text at a particular pixel position within the window. This allows great flexibility in positioning characters of different fonts and graphics within the same window, but in turn it makes it more complicated to program simple things such as the common text window -- scrolling in itself requires a call to bit-copy routines to copy the current display up a few pixels and clear the last few rows of pixels. The common pop-up menu is created using a window that contains several **sub-windows**, where each item in the menu is a separate sub-window, and chosing a menu item corresponds to receiving a XButtonPress or XButtonRelease event within the window for the item chosen. Keyboard input is similarly complicated in that it relies on the primitive operations of XKeyPress and XKeyRelease events.

The act of programming an Xlib application in itself is a long and arduous process. In order to just get an empty window open, the program must first open a connection to the display, then retrieve the screen pointer that it will need for that display (one display connection may have multiple screens in a multiple screen workstation), then set up the parameters for the desired window (including size, color and various other attributes), create the window in memory, and then map the window onto the screen (to make it appear), making sure it's raised (not obscured by other windows). Input is more foreign: the desired types of input to the window must be selected explicitly and then all input events to the application's windows of the desired types (including keypress, mouse button press, mouse movement, expose region, window resizing alert, etc.) are received through events which the application must be prepared to process appropriately at any given time.

The **expose** event is especially foreign to those of us not familiar with the concept of being responsible for redrawing the screen or parts of it. For graphics applications, this might even mean keeping a spare copy of the output window stored somewhere in memory, so that at any point, the exposed regions can be easily redrawn. But this is rather memory inefficient. There is not nearly enough room here to discuss all of the different techniques for event handling. Suffice to say that it is a topic that books have been written on, and that it is an art unto itself.

In allowing flexibility, X and the Xlib interface not only make the task of programming more complicated, but also allow enough variations on the user interface with the windowing system to be quite confusing to the user. To both make the task of programming easier and to allow a more standardized interface to the environment, tools have been built, and libraries have been built on top of the basic Xlib interface to collect common objects into more coherent groups, and handle common windowing tasks without the programmer having to re-write the wheel.

CASE Tools: Window Editors

CASE (Computer Aided Software Engineering) tools, are emerging as the answer to the problems of creating visual displays "blind". Although specifying screen coordinates may be sufficient for simple text interfaces, graphical interfaces need more precision. Screens for this type of interface are "drawn" more that just laid out. The tool that is needed is one that allows the user not only to create a simple interface, but to create the interface in a simple manner, so that as little time as possible is spent in changing coordinates and position as the programmer tries to get the screen to look "just so."

One approach to solving this problem of complex graphic layout is to produce tools which can aid the programmer in developing the basics of a windowing application. CASE tools have been gaining in popularity in recent years, especially in more graphical fields such as this. By using another program to design the application, the programmer can be freed from such tasks as the fundamentals of event handling, and concentrate on the functionality of the program and the layout and interface of the screen.

One (hypothetical) example[2] of such a program is "WE" the "window editor". "WE" is capable of generating a "blank window" on which the programmer can add menus, buttons and subwindows, using construction-set techniques. Then, with a button selection, WE generates stand-alone C code with Xlib[3] calls to setup that window, and handle screen refreshes, allowing the programmer a "switch" statement from which to dispatch routines, when a button is pressed onscreen.

When the program is started, it presents the programmer with a small blank window, and with a menu of three generic objects to put in that window:

- The canonical simple, generic subwindow (for use in text or graphic displays).

- Pull-down menus patterned after home-computer type menus.

- "Buttons" used for making selections with the mouse.

Using the window manager interactions, the programmer may then resize the output window and reposition it on the workstation screen. Using the menu selections, the desired program window can be created, using the mouse to position and reposition buttons, specify the areas that subwindows cover, or creating the necessary pull-down menus by entering the desired text.

Once the created window appears as desired, the programmer clicks on the "code" menu option, causing WE to look through its collections of window parameters and formulate the C and Xlib code necessary to generate the desired input/output window. Programs generated by WE are usually about 1500 lines long.

[2] Although I actually created this program as a project some time ago, it was independent of any corporation, and not intended for release. However, such programs do exist on the open market, and provide much more robust functionality.

[3] Although the use of toolkits would considerably shorten the length of the programs, programming in Xlib has the advantage of allowing the programmer to be as flexible as they desire with the program's implementation, regardless of current user interface fashion or standards.

Although these programs are relatively simple in their structure, they are still quite a bit of code, primarily because they are written at the low level of the Xlib interface. Instead of re-writing the basic code as Xlib functions in each program, many have combined these functions into libraries, not only allowing a faster start-up time for code generation, but giving a uniform interface to all applications written using the library.

The Toolkit Approach

Libraries of common X functions are usually referred to as "toolkits", patterned after the XToolkit, one of the first to offer higher-level interfacing with Xlib. Using a toolkit, programmers can concentrate on the "meat" of the code instead of worrying about all the low-level interactions with the windowing system. By conforming the style of the interface, toolkits allow simple interactions to create otherwise cumbersome features. Toolkits provide ways to build and display commonly-used interface devices such as pop-up menus, and "buttons" which respond in a uniform way to user interaction. Toolkits also implement general algorithms for handling expose events, so that the programmer doesn't need to be concerned with generating expose-handlers for each application.

Unfortunately, in simplifying the style of interface, toolkits run the risk too much restriction, deviating from the principle of flexibility that X was based on. Also, although most applications could just work around unneeded parts of the toolkit, programs written for simple applications using the Xlib calls directly, would run faster than the same application would using a toolkit, due to the toolkit's sheer complexity. There will always be those who still prefer to do things the harder, faster way.

Motif, a particular style of interface, is gaining wide acceptance as a windowing interface standard to X11. HP's OSF/Motif package contains pre-defined sets of "widgets": several different buttons, menus, scroll bars, text windows, etc, all of which have a similar "3-dimentional" appearance, and allow many interactions that are independent from the mouse. Many menu selections and buttons may be chosen either with or without the mouse, allowing users to speed up their use of the application once they become familiar with it. Motif is a library of specific types of widgets, built on top of the XtIntrinsics library (allows for the creation of many different types of widgets), which in turn builds on Xlib. By defining this standard interface, uniform applications can be produced with much less time building up the basics of the windowing interface.

Getting It All Working

The HP 9000 / Series 800 allows different types of high-resolution graphic displays to be used with X11, as well as multiple displays on the same system, and multiple screens with each display. The X server is started on a particular display on a system using the **x11start** command, with the **DISPLAY** environment variable set to indicate which display the server is to be using. The format of the DISPLAY variable in X11 is "host:display.screen", with *host* as the machine's name (defaults to local), *display* the display number (defaults to 0), and *screen* the screen number within that display (defaults to 0). The x11start program uses the display number to retrieve the configuration of the X server from the file /usr/lib/X11/X*n*screens, where *n* is the display number specified in the DISPLAY variable. The X*n*screens file contains server configuration information as to how the different screens within that server are accessed, the number of image planes on the devices, how the planes are divided, and so on.[4]

Once the server(s) are started on the display device(s), applications may then use the same display specification to indicate that they wish to use that particular display for input and output. Applications will usually reference the DISPLAY environment variable or an explicit *display* argument to specify which X server to use. By merely changing the host name in the display string, an application can run as easily on a local server as on another machine's server across the network, a functionality that is transparent to applications written within the X11 specification. This access to a remote server is controlled through the xhost(1) "X window system access control program" and the /etc/X0.hosts file, both residing locally on the X server's host machine. By conforming to the X11 protocol, any application can be run within the X specification can be displayed on any X11 server, independent of make.[5]

When the server is started by a user, the *xinit* program checks that user's home directory for a file named **.x11start.** If it exists, commands in that file are executed as a shell script after the X server is started.[6] If it doesn't exist, xinit will instead use the script found in /usr/lib/X11/sys.x11start. This file usually starts initial applications like the window manager and terminal emulator applications to communicate with the local host.

Many X applications read standard configuration information from **defaults** files on the machine from which they run, replacing the need for excessive numbers of

[4] The details of configuring an X11 server can be found in *Using the X Window System, HP 9000 / Series 300/800 Computers,* and will not be discussed in depth here.

[5] This does not imply that a compiled X application can be *run* on any platform. Executables are still machine-dependent, while Xlib source is relatively portable. The point is that an X11 application running on one platform, while still *executing* on that platform, will be able to use any X11 server as its *display.*

[6] This is similar to the idea of "rc" files common to other UNIX applications. Files like .mwmrc are read by the application (in this case mwm, the motif window manager) after they start up, and allow the user to customize applications' startup behavior. In the case of .mwmrc, after the motif window manager is started, it reads this file to determine what menus the user wants, how it is to react to buttons pressed on the mouse (ButtonBindings) and keys on the keyboard (KeyBindings), as well as other client-specifics.

command-line arguments. These configuration files specify desired parameters such as size, color, fonts, and application behavior, depending on the application's needs, and are usually documented with the particular application. By default, startup values are pre-defined in configuration files under /usr/lib/X11/app-defaults/*application*. For customization, default information about an application can be appended to a user's **.Xdefaults** file (also in the home directory of that user). The lines in these default files have the format:

```
application-name*resource:  value
```

The application-name part of this string is determined by the filename of the application, allowing the same application to be run with different defaults merely by linking that application to different names.

The window manager[7] is able to customize its behavior relative to different classes of applications running on the same display. It determines the application's identity through an X protocol request to the new window, which will respond with information about itself, if that information has been provided to the server.[8] The window manager can then associate application-specific behavior to that particular window, as configured in its own defaults file. Note that the window manager, as a regular application, only has access to the defaults files on the machine that is running it, not to the machine that is running all applications on that display. This means that if your window manager is started on machine A, it will only get information from the X server and from defaults files on machine A. If an application is run on machine B, it will read the files on machine B to customize its behavior, but the window manager will still be reading the files on machine A to determine how to adjust its client-specific behavior. Window manager specifics (such as icon bitmaps, for example) must appear on the same machine that runs the window manager, and refer to other applications only by their internal names, not filenames.

Machine Independence and Resources

When the X server and the application are running on separate machines, people are often confused about what the application is reading when, and for what. One of the most common misconceptions about X is that the fonts -- which describe the bit patterns that make up alphanumeric characters and are used by applications for

[7] Actually, any X application has the ability to access this information through the same requests that the window manager uses. It is just much more common for the window manager to be accessing this type of information than it is for other types of applications.

[8] Since this identification information is provided in the application's code itself, it is not modifiable by merely changing the applications filename, although it is possible for the application programmers to allow users to optionally change the application's identity, if they so desire. Many, but not all, applications adhere to a convention that defines their internal names to be their filenames with the first two letters capitalized (eg. XClock, XLock).

displaying text -- are read on the machine that is running the application. Actually, they are a resource that is part of the X server. Among the server's internal tables of parameters is a font path. An application makes a request for a particular font *resource* by name, and the server determines if that font is available or not, returning relevant font information to the application. An application may request that the server change its font path, so a particular font may be found: one application that sets several server-specific parameters is xset(1). Other operational parameters that are stored on the server which xset manipulates include bell parameters, keyclick, and mouse acceleration.

Usability

Considering the code necessary to generate a seemingly simple application (at least 1500 lines or more if programmed in Xlib directly), and the complexity of the application environment, one might ask if windowing environments are at all worth the effort. The simple fact is that, although cumbersome in ways, the windowing environment is a great improvement over the older style of purely textual interface. To be fair, one should also note that the concept of the windowing interface is still quite new, and X itself (the first and currently only *standard* interface package) is just a few years old (with X11 having just been introduced just over two years ago). As tools and libraries develop, the task of programming under X will undoubtedly become less cumbersome.

Having been developed with openness in mind, the X specification is very receptive to advancements in the industry, and as new ideas emerge, they will undoubtedly incorporated into the X standard. At different times each year, X users and developers gather in various forums to discuss developments in X, as well as X's direction. Through conferences like Xhibition and the MIT consortium, X will continue to develop, remaining current and responsive to the windowing community's needs.

Verifying PC Assembly Data using Computer Graphics

Jay Shain
Philip Walden

Hewlett-Packard
3155 Porter Dr., Palo Alto, CA 94304

1. Overview

The Neutral-File Graphics System (NGS) was devised to reduce the disruption of Hewlett-Packard (HP) SMT (Surface Mount Technology) assembly processes due to errors in assembly data derived from product design data. Prior to the deployment of NGS, the primary method of verifying design assembly data was by visual scanning of data files or by use of assembly machinery. In fact, some types of assembly data errors were attributed to poor machine calibration. It was only after NGS was deployed that the errors were discovered.

2. R&D to SMT Assembly Process

Most SMT assembly centers at HP accept prototyping and production jobs from a variety of design sites. These sites use a variety of CAD[1] layout methods and systems. This mix of systems complicates the process of linking design systems to factory floor equipment. This process is also a major source of assembly errors.

2.1 Data Flow Diagram without NGS

In order to link several types of CAD layout systems to factory floor equipment, most SMT assembly centers use CAD-DLS [1] (CAD Data Link System) based on the HP9000 HP-UX platform. CAD-DLS was developed for use within HP and uses a set of "Neutral Files" as a common point of exchange between design sites and assembly sites. Figure 1 illustrates the process. Processes that convert CAD specific data formats to neutral files are called translators. Processes that convert neutral files to assembly machine instructions are called recipe generators. Using a neutral file mechanism simplifies the task of linking multiple types of design systems to several types of assembly equipment, from a geometric problem to a linear problem. Thus, supporting a new CAD layout system involves developing just one new translator. Supporting one new placement machine requires developing just one new recipe generator.

2.2 Sources of Error

The above process has two main sources of error. The first category are operator errors injected by designers, engineers or technicians. These sources include misinterpreting standards, misinterpreting documents and instructions, and of course typos. The second category of errors are systematic and have to do with the robustness of the translators and generators.

2.2.1 Operator Errors

Figure 1 identifies the major sources of operator error in the CAD-DLS system. These include subpanel board-image locations, CAD-origin offsets, IRM,[2] location errors, tooling hole locations, non-standard component libraries and material list errors.

2.2.2 Systematic Errors

Figure 1 also identifies the sources of systematic errors. These are almost exclusively in the translators. Translators typically make some assumptions about the definition and standards of the

[1] CAD refers to computer aided printed circuit design systems.

[2] Image Reject Mark: a small .25 inch square area on a PCB used to mark bad board-images on a multi-image panel. Automatic assembly machines detect marked IRMs and do not place parts on the bad image.

incoming CAD design data. If the design data does not follow the standards or uses a different definition then errors may result. For example, the EGS[3] translator assumes that designer using EGS would use the EGS Net List program to verify design connectivity. To use the EGS Net Lister, the designer must define library parts with electrical ports and follow part library conventions defined in the EGS documentation. The EGS translator takes advantage of the same rules to determine which side of a board a part is to be placed. If the designer hand-checks connectivity via visual inspection and doesn't define his or her parts with the net list conventions, parts may be placed on the wrong side.

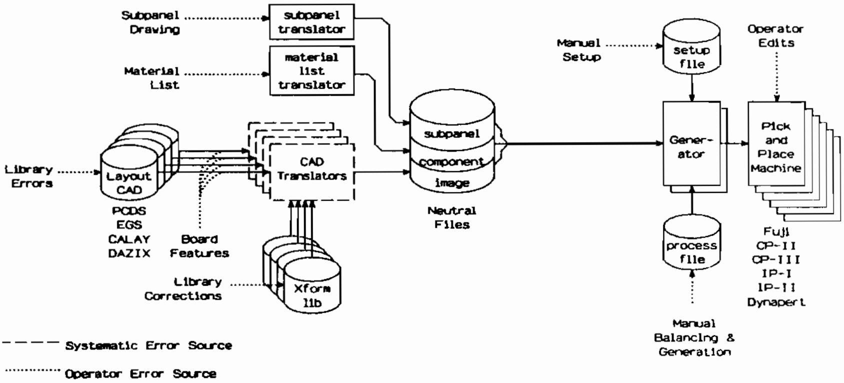


Figure 1. CAD-DLS Data Flow

Some translators are better than others. The PCDS[4] translator makes the fewest assumptions and thus has the lowest systematic error rate. Generally, operator errors are the greatest source of problems and are typically the hardest ones to detect.

2.3 Cost of errors

Without the ability to easily detect assembly data errors before they reach the production line, the cost of such errors can be greatly magnified. Using production equipment to simulate and verify the assembly data has the following costs.

- Additional process engineering or technician time to generate correct recipes.
- Delays in prototype turn-around.
- Loss of production line time.
- Lower machine utilization.
- Higher costs per placement.
- Lower production capacity.
- Lost revenue.

[3] HP's Engineering Graphics System

[4] HP's Printed Circuit Design System.

3. Proposed Solution

NGS was developed as a highly focused solution to detect and verify the correctness of assembly data before it hit the production line. The concept was to provide a rapid method to locate errors and verify correctness without using production equipment.

Since most of the data was geometric in nature, the use of computer graphics to render the data human-readable was a logical solution. NGS allows the user to quickly generate a graphical representation of the panelized assembly. Gross errors, such as missing or overlapping parts, are immediately obvious. Comparing the output of NGS to assembly drawings from the CAD system locates part rotation errors. Lastly, NGS can be used to generate 1:1 scaled plots for direct overlay comparison with bare panels and stencils to find even small alignment/offset problems.

NGS also allows users to verify process plans by color coding of parts by machine responsibility.

Finally, as NGS was not intended to actually remove the sources of error, it was developed with minimal resources and time. Thus remaining resources could be properly applied to improved translators that actually reduce systematic and operator error sources.

3.1 Data Flow with NGS

Figure 2 positions NGS within the CAD-DLS system. Note that all assembly data comes from the neutral files. Specifically subpanel, component (material list) and board-image data is used. This data includes board-image locations in the subpanel, component locations, tooling holes, fiducials,[5] IRMs, reference designators[6] and footprint names.

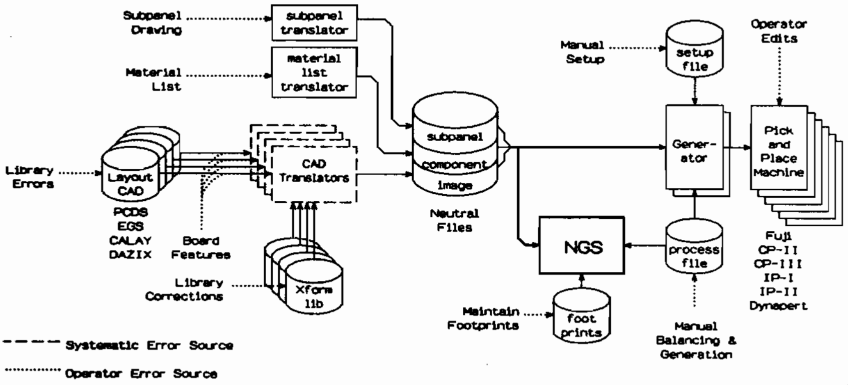


Figure 2. CAD-DLS with NGS

Machine responsibility information comes from the process files setup by process engineering.

The only error sources not covered by NGS are setups and edits made by operators to recipes loaded in machines.

[5] A fiducial is a small .060 inch circular PCB pad used for precision registration of PCBs on automatic assembly machines.

[6] A reference designator is a unique identifier for a component on a PCB. Usually the first character is a letter signifying the component type. For example, U504 typically is an integrated circuit.

Also, note that the actual geometric definitions of component footprints comes from a directory of footprint files, not from the neutral files. The user must maintain this directory and has the task of ensuring that the directory contains footprints used by the assembly data in the neutral file. This requirement is the most time consuming part of maintaining the NGS system.

3.2 NGS Implementation

NGS leverages a graphics package, called SKETCH. [2] SKETCH uses the Starbase graphics library which allows for a device independent graphics strategy.

3.2.1 Leverage of Existing Code

By leveraging SKETCH subroutines, a working framework for NGS was developed in several days. Another three months was spent working with the customers on enhancing the features, writing documentation to make it easily installable and usable. In addition, a prototype version of a CAD-DLS user interface was also leveraged to make everything menu driven.

3.2.2 Multiple Device Support

Due to the large variety of graphics devices found in HP factories, it is important that NGS support as many of them as possible without overwhelming the support group. Because SKETCH is written using device independent Starbase,[7] as long as Starbase supports the device, NGS can output a drawing to that graphics device. NGS display outputs are supported on graphics terminals and PC terminal emulators (2397A and 2627A families), X11 windows, and bit-mapped displays not running windows.

NGS supports hard copy outputs on the Draftmaster I and II, PaintJet, LaserJet, and 7550A plotters. The user chooses the appropriate Starbase device driver for the plotter. These drivers are normally the "hpgl" or "CADplt" Starbase drivers. Drawings are easily scaled and rotated by selecting the right parameters inside the menuing system.

3.2.3 Graphical Hierarchy and Implementation Structure

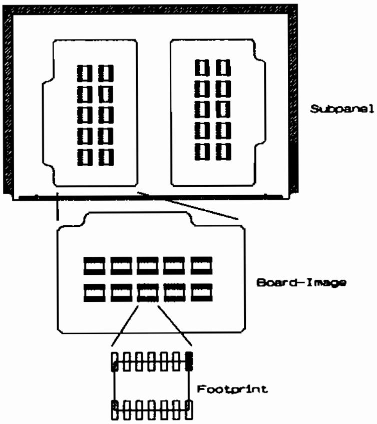


Figure 3. Hierarchy of NGS Graphics Objects

[7] HP's implementation of the ANSI CGI (computer graphics interface) standard.

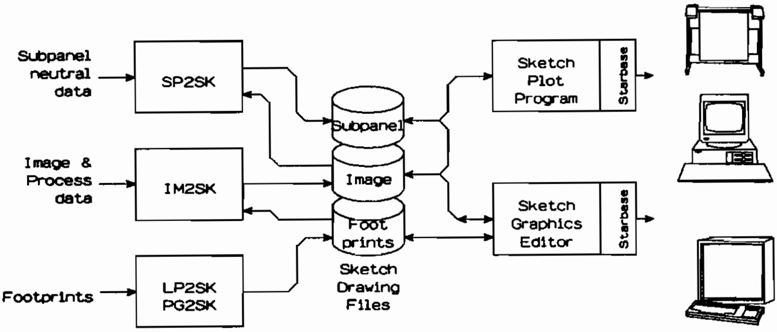


Figure 4. Neutral-File Graphics System Structure

There are three types of SKETCH drawings created using NGS: footprints, image and subpanel as shown in Figures 3 and 4. These three types constitute a hierarchy of graphics objects. Footprints are parts on an image and images are parts on a subpanel.

Each site has to maintain a directory of footprint drawings, which are outlines of either surface mount solder pads or through-hole finished holes on the board. NGS provides the programs, lp2sk and pg2sk (land pattern to sketch and pad geometry to sketch respectively) which create footprint drawings based on location and size of the individual solder pads or location and diameter of the finished holes. These programs take as input a flat data file that contains the coordinates and sizes of the actual lands or pads.

When an image drawing is created, the program, im2sk (image to SKETCH), looks for the associated footprint name in the footprint directory. If it is found, the footprint drawing, along with the reference designator of the part, is included in the image drawing. If it is not found, the reference designator is only included. Each footprint is then put under a certain machine responsibility group. Finally, image fiducials and tooling holes, image reject marks, reference designators, and the image origin are added to the image drawing.

By editing a configuration file the user is allowed to include or exclude any machine responsibility groups and plot that groups of footprints in a specified color. A flat data file known as the responsibility file, which maps which parts get loaded by which machines, is compiled into the image information. Figure 3 refers to this type of input as process data. The color of pin number one for a part's footprint is also configurable. Parts that are not to be loaded for a particular machine can be plotted in a configurable color. With this flexibility users are able make colorized inspection drawings and documentation for the pick-and-place machines and the handload and backload areas.

The program, sp2sk (subpanel to SKETCH), steps and repeats the image information at the proper location and orientation. The image drawing itself is created by calling im2sk. A standard half or third subpanel border, subpanel fiducials and tooling holes, and image origin markings are added to complete the subpanel drawing.

Once drawings are created, the SKETCH graphics editor can be used to edit the drawings to add annotations or modify the results. Parts with complex shapes, such as connectors, can be easily created with the graphics editor. Using SKETCH to edit the drawings makes it easy to purge out different machine responsibility groups and to customize the drawing to fit the need.

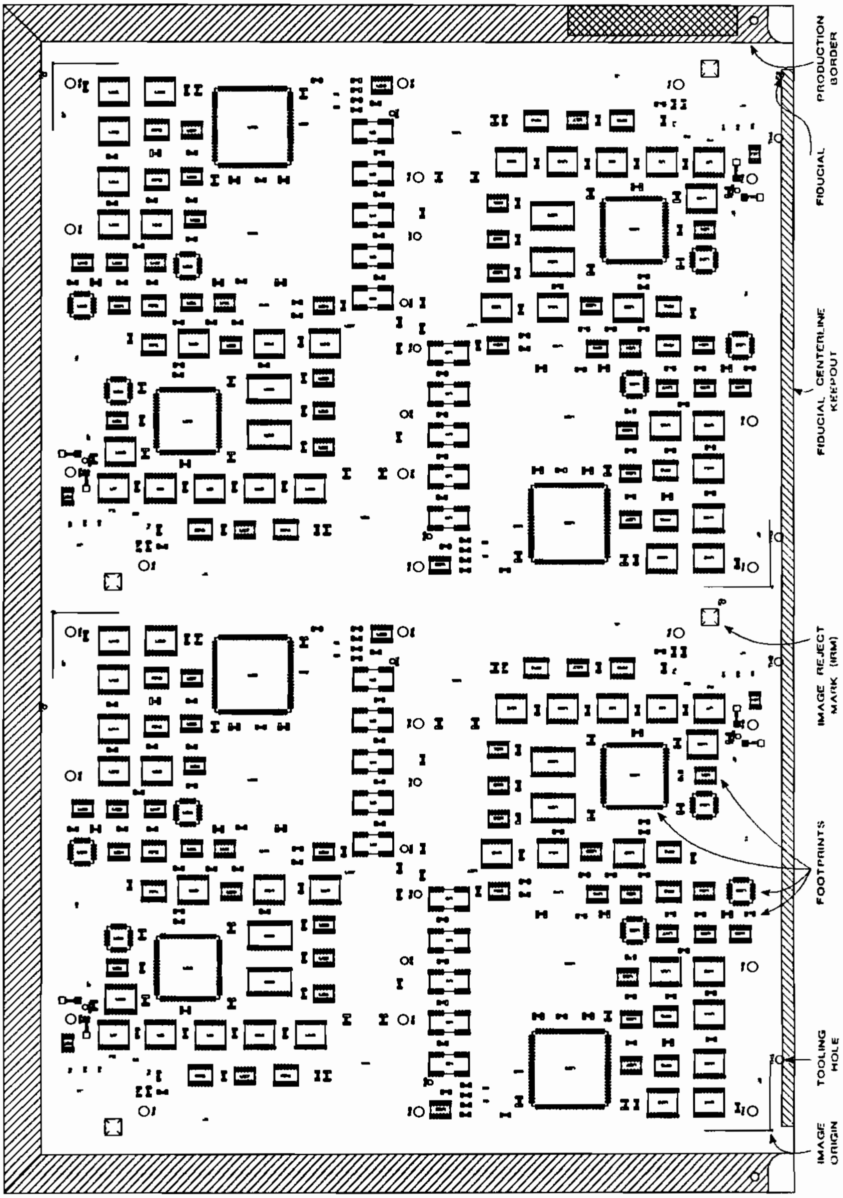


Figure 5. Example of NGS Half-Subpanel Drawing

3.3 NGS Operation

3.3.1 Examples of Output

Figure 5 illustrates an actual NGS subpanel drawing overlay for verifying neutral data against the bare subpanel board or verifying solder paste stencils. Enlarged image drawings can be used for handload/backload documentation.

3.3.2 Examples of Site Usage

NGS was originally intended for the sole purpose of verifying the integrity of CAD-DLS neutral files. Since its release its use has been expanded to inspection, handload, and backload documentation and solder paste stencil verification.

3.3.2.1 Off-line Verification of Neutral Data NGS is used to verify the neutral data, which is used to generate recipes for manufacturing pick-and-place machines, without having to generate a recipe or actually utilize machine time to place parts. 1:1 scaled overlay drawings are created on transparent media and compared by "overlaying" the drawing on the actual bare subpanel PCB.

Recently, designers who have been using CAD-DLS translators have been using NGS to verify the neutral data before shipping it to the surface mount site.

3.3.2.2 Documentation Some Surface Mount Sites create colorized NGS drawings for inspection documentation for each of their pick-and-place areas, handload stations, and backload stations.

3.3.2.3 Stencil Verification The HP Product Generation Processes Prototype Lab even uses NGS to verify their solder paste stencils.

3.4 Errors Detected

As the users have discovered, NGS drawing outputs do not lie. If there is something incorrect with one of the neutral files, NGS illuminates the problem. Below are the types of errors that have been detected at the various surface mount sites and design sites.

- location and/or rotation errors
  - image origin
  - components
  - subpanel and image image reject marks
  - subpanel and image fiducials
  - subpanel and image tooling holes
- incorrect subpanel and/or image dimensions
- incorrect subpanel or image units
- missing parts
- incorrect part machine responsibility
- falsely loaded parts

3.5 Benefits of NGS

The largest benefit of NGS is that it gives quick, simple, inexpensive feedback about the integrity of the neutral data without using valuable pick-and-place machine time. By creating a drawing which is overlaid onto a subpanel, errors are more easily noticed and pin-pointed. Using a pick-and-place machine is costly and only introduces extra variables into the picture, such as pick-and-place placement accuracy problems. The process of verifying the neutral data, which could take up to several days, now takes only minutes to a few hours.

Once configured for the site, NGS can be used by an operator or software technician because of its easy user interface. Feedback is graphical and it very simple to understand the inconsistencies of the data. By freeing engineering resources, which are scarce on the manufacturing floor, the new product introduction process is shortened.

Another benefit of NGS is early and accurate detection of errors. By having board designers generate their own neutral files using a CAD-DLS translator for their specific design system, NGS can help verify the integrity of the data where it can most easily be corrected.

REFERENCES

1. Lujack, J. and Safai, F.: **CAD Data Link System,**
   Proceedings of the Technical Program NEPCON East '89 at Boston,
   Cahners Expo Group, 1350 E. Touhy Ave., Des Plaines, IL 60017, June 1989, p. 15.
2. Walden, Philip C.: **The Design of Sketch, A General-purpose Graphics Editor,**
   Proceedings of the 1985 Conference at Washington D.C. HP1000 and HP9000,
   Interex, 680 Almanor Ave., Sunnyvale, CA 94086-3513. Sept. 1985, p. 1013.

A Data Acquisition Application using HP BASIC-UX and the SAS^{tm} System

John Hall

Hewlett Packard Company
Raleigh, NC

Abstract

The excellent data acquisition environment of HP BASIC-UX can be enhanced by the data reduction, analysis, and presentation capabilities of the SAS^{tm} system. This paper will address the technical application of temperature measurement from a data acquisition device controlled by HP BASIC-UX. The entire process is monitored by the SAS system, which performs analysis of the data and presents the resulting chart in a 3D graphical representation.

Introduction

This application was designed and written to demonstrate the integration of the SAS system into a data acquisition environment. The capability is demonstrated to effectively 'control' an external process - in this case, data acquisition from an instrument. The SAS program 'launches' the data acquisition process (which executes in HP BASIC-UX) with the SAS program 'suspending', waiting for the completion of the data acquisition. Control is returned to the SAS program, which does an analysis on the data and produces a 3D smoothed grid plot, using temperature probe number, time, and actual measured temperature as the X, Y and Z axes, respectively.

The SAS System

The SAS system provides the HP-UX workstation user with the most needed application solutions - superior data analysis, efficient data management, and powerful reports and graphs. Complex multivariate techniques as well as tables, charts, 3D plots and prism maps can be generated from menu selections, making even first-time users productive. For complete application development and prototyping, the SAS system's fourth-generation language works in conjunction with a network of integrated products. For this application, source code (shown in Appendix A) was developed and submitted via the SAS Display Manager, a multi-windowed prototype environment.

HP BASIC-UX

HP BASIC has long been the language of choice for instrument control. In fact, it was created by engineers to assist in the complex task of developing test systems. HP BASIC has a powerful command language, a highly optimized program development environment, and exceptional device I/O. When used with a HP-UX workstation, HP BASIC-UX provides the HP-UX user with the multitasking and multiuser capabilities while retaining HP BASIC's superior I/O capabilities.

Technical Data

The data acquisition unit used for the demonstration was a HP3852A data acquisition and control system. This highly expandable test system can be configured with over 30 different plug-in modules, including actuators, counters, analog outputs, and digital I/O. For this demonstration, a HP 44708A 20-channel relay multiplexer interface was used to interface the thermistors to the HP3852A. The thermistors were connected via cables to the 44708A card, allowing visitors at the demonstration to handle the thermistors during the data collection, thereby causing temperature fluctuations and a more interesting graph. The HP3852A was connected to the HP 9000 series 360 workstation via HP-IB.

Application software included the SAS system, HP BASIC-UX, and X11. The SAS program was executed from within a HPTERM window in X11, with HP BASIC-UX opening its own window for execution. The resulting graph was displayed in a graphics window on the X display. Through the SAS system's support of the Starbase Graphics library, the graphical output from a SAS program can be displayed in a Starbase-generated graphics window.

The SAS system is 'step-oriented', with one step or procedure executing to completion before the execution of the next procedure block. This distinction works well when interfacing the SAS system with companion software like HP BASIC-UX. A simple BASIC program, shown in Appendix B, instructs the HP3852A to collect temperatures from five different probes for approximately 45 seconds. During this collection period, the temperature from each probe is displayed in the HP BASIC-UX window on the HP workstation.

The SAS program which runs the entire process is shown in Appendix A. This program instructs the SAS system to establish links to files via the FILENAME statement (line 1) and sets certain options for the SAS graphics (line 2). The SAS program next executes HP BASIC-UX, providing the name of the source program in the runstring (line 3). This causes the HP BASIC-UX program window to open on the workstation display and begin execution. During the execution of the HP BASIC-UX program the SAS program is suspended, waiting for the completion of the BASIC program.

The BASIC program , shown in Appendix B, is quite straightforward, identifying the type of thermistor in each channel of the data acquisition unit (lines 1140-1190). The program instructs the HP 3852A to begin scanning the probes for measurements. The results are both displayed on the screen and placed in a file for later use by SAS. The collection process continues approximately every three seconds for 45 seconds. It should be noted that 45 seconds was chosen as a arbitrary value for demonstration purposes, and any value could be used in an actual application. After the data acquisition portion has completed, the HP BASIC-UX window closes and control is returned to the SAS program.

The SAS program builds a dataset and assigns labels to the variables. A simple calculation is preformed to convert the temperatures, read in Celsius, into Fahrenheit. The SAS program next produces a 3D grid plot in the Starbase graphics window (Appendix C). All aspects of the graph, including smoothing factor, axis granularity, rotation, and tilt factors can be defined in the SAS program. Other options may be selected that were not pertinent to this demonstration.

Summary

This demonstration very effectively shows the ability of complementary products to enhance the capabilities delivered to the HP-UX workstation user. Through exploiting techniques such as these, HP-UX workstation users can greatly increase their overall effectiveness, thereby increasing their productivity and efficiency.

Acknowledgments

The author wishes to acknowledge Bob Moore and George Culbertson, whose guidance, direction, and support helped make the project possible. I would also like to thank the Marketing and Technical Support departments at SAS Institute for their continued support.

Appendix A: SAS source code program - tempmeas.sas

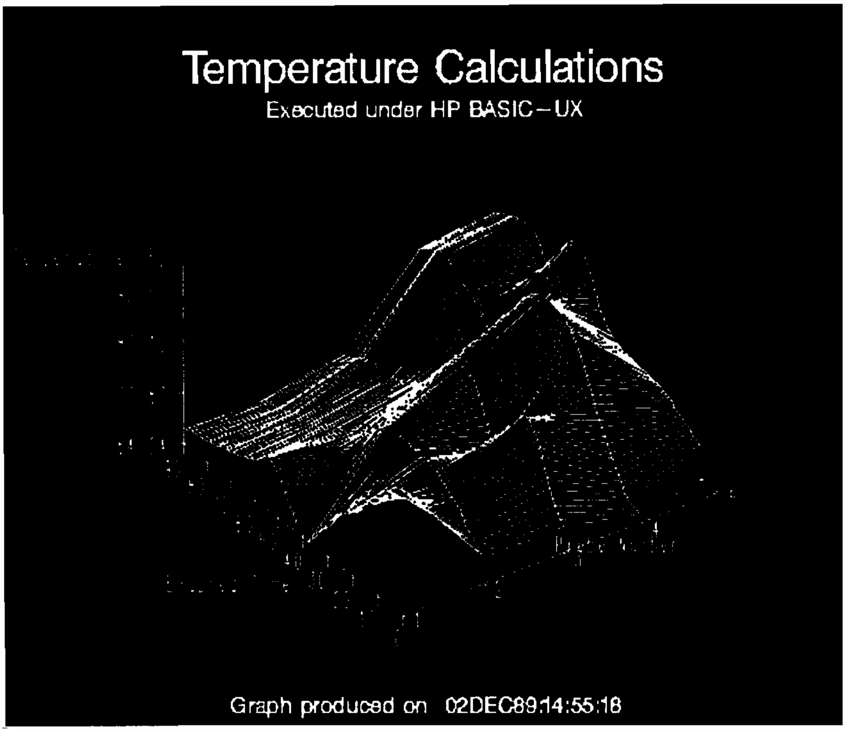
```
goptions dev=hpcat noprompt ;
x ’rmb TEMP.BAS’;
filename d "./HFILE" ;
data john;
  infile d;
  input y x z 8.2;
  z=((9/5 * z) + 32);
    label y = ’Probe Number’;
    label x = ’Elapsed Time’;
    label z = ’Temperature (F)’;
run;
data _null_;
 today=datetime();
proc g3grid data=john out=johnout;
  grid x*y=z/
  naxis1=100
  spline
  smooth=.05
  axis1=0 to 30 by 1
  axis2=1 to 5 by 1;
run;

filename new ’/users/sas/new.dat’;
data _null_;
 now=datetime();
 file new;
 put ’title1 f=swiss ”Temperature Calculations”;’;
 put ’title2 f=swiss ”Executed under HP BASIC-UX”;’;
 put ’footnote1 f=swiss ”Graph produced on ’ now datetime17.’”;’;
run;
proc g3d data=johnout;
 plot x*y=z/ caxis    =  white
          ctext    =  white
          ctop     =  yellow
          cbottom  =  cyan
          xticknum =  5
          yticknum =  10
          rotate   =  35
          tilt     =  40;
    %include new;
run;
```

Appendix B: BASIC program TEMP.BAS

```
1000 OPTION BASE 0
1010 DIM Tdata(29,4),Tpts(4)
1020 INTEGER Complete,Done,Ctr,I,J
1050 ASSIGN @Hp3852a TO 709
1060 OUTPUT @Hp3852a;"RST"
1080 REPEAT
1090 UNTIL BIT(SPOLL(@Hp3852a),4)
1100 !
1110 OUTPUT @Hp3852a;"SUB RDR;"
1120 OUTPUT @Hp3852a;"  REAL D(4);"
1130 OUTPUT @Hp3852a;"  DISP OFF;"
1140 OUTPUT @Hp3852a;"  CONF MEAS TEMPT 105 INTO D(0);"
1150 OUTPUT @Hp3852a;"  CONF MEAS TEMPK 101 INTO D(1);"
1151 OUTPUT @Hp3852a;"  CONF MEAS TEMPJ 100 INTO D(2);"
1170 OUTPUT @Hp3852a;"  CONF MEAS TEMPK 104 INTO D(3);"
1180 OUTPUT @Hp3852a;"  DISP ON;"
1190 OUTPUT @Hp3852a;"  CONF MEAS TEMPK 103 INTO D(4);"
1200 OUTPUT @Hp3852a;"SUBEND"
1240 Done=0
1290   OUTPUT CRT
1300   OUTPUT CRT
1310   Complete=0
1320   Ctr=0
1340   ON CYCLE 1.5 GOSUB Gt
1370 Wt:IF NOT Complete THEN GOTO Wt
1380   OFF CYCLE
1450 New:ASSIGN @F TO "HFILE"
1480   FOR I=0 TO 29
1490     FOR J=0 TO 4
1500       OUTPUT @F USING "D,3X,DD.D,3X,DD.DDDDD";J+1,I*1.5,Tdata(I,J)
1510     NEXT J
1520   NEXT I
1530   ASSIGN @F TO *
1580 QUIT
1600 Gt: !
1610 OUTPUT @Hp3852a;"CALL RDR"
1620 OUTPUT @Hp3852a;"VREAD D"
1630 ENTER @Hp3852a;Tpts(*)
1640 MAT Tdata(Ctr,0:4)= Tpts(0:4)
1660 FOR I=0 TO 4
1670   IF I<4 THEN
1680     OUTPUT CRT USING "#,DD.DDDDD,4X";Tpts(I)
1690   ELSE
1700     OUTPUT CRT USING "DD.DDDDD";Tpts(I)
```

```
1710   END IF
1720 NEXT I
1750 Ctr=Ctr+1
1760 IF Ctr>29 THEN
1770   Complete=1
1780   OUTPUT CRT
1790   OUTPUT CRT
1800 END IF
1810 RETURN
1830 END
```

Appendix C: Sample output



INTEGRATING HP-UX AND MPE-XL SYSTEMS IN A LAN ENVIRONMENT

by

Daniel R. Kaplan

Industrial Applications Center
Hewlett-Packard Company
1266 Kifer Road
Sunnyvale, California 94086

Abstract

Since the capability of physically linking MPE-XL and HP-UX systems through a network is widely available, the task of integrating computer hardware is mainly an issue of integrating the software applications that reside on the different computers. HP-UX and MPE-XL computers are increasingly found operating together in the same environment. This paper discusses the problems of integrating software applications residing on HP-UX computers with software applications residing on MPE-XL computers, and is applicable in heterogeneous environments generally.

Integration problems are analyzed and categorized into areas of Data Access, Data Transport, and Data Compatibility. Each of these categories is further broken down into component issues which are discussed as to cause and effects. Finally, strategies of minimizing or eliminating these problems are suggested, and discussed with respect to implementation considerations.

Introduction

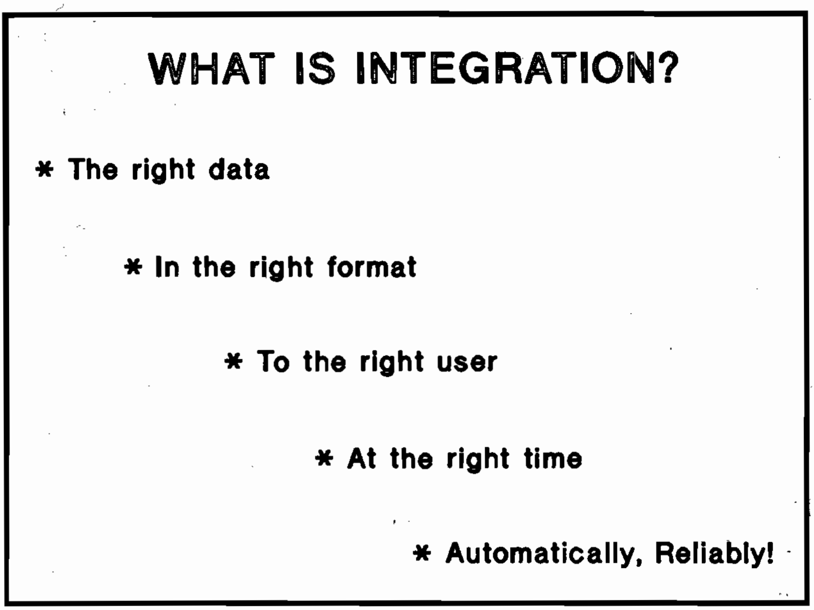
MPE-XL and HP-UX computers are increasingly found operating together in the same environment, since a smoothly functioning manufacturing business, or other type of enterprise, requires features that each operating system is quite good at. The capability of physically connecting HP-UX and MPE-XL systems through a local area network (LAN) is widely available today. Yet a LAN is only a piece of the solution to achieve true integration in this heterogeneous environment.

To a large extent, the task of integrating computer hardware is really an issue of integrating the software applications that are the sole purpose for the existence of the hardware in the first place. This paper is offered as a discussion of the problems of integrating software residing on HP-UX computers with software on MPE-XL computers. Much of the discussion is applicable to more generally heterogeneous environments.

First, integration is defined in terms of the desired benefits, and the general problems that are encountered in integrating software applications in a heterogeneous environment are discussed. These problems are briefly analyzed and categorized into areas of data access, data transport, and data compatibility.

Since it is commonly asserted that file transfer can be used as a mechanism to integrate software applications, file transfer will be discussed as a necessary but not sufficient component of true integration. Then the functional requirements and supporting infrastucture of application to application interfacing are discussed, followed by detailing those problems specific to integrating HP-UX and MPE-XL systems.

Finally, a solution is proposed and HP's efforts to make this solution available to its MPE-XL and HP-UX customers are explained. HP Software Integration Sockets (HP Sockets) is introduced and described as a potential solution to the problems of integrating software applications running on MPE-XL with applications running under HP-UX.



WHAT IS INTEGRATION?

* The right data

* In the right format

* To the right user

* At the right time

* Automatically, Reliably!

In its simplest expression, integration is getting the right data in the right format to the right user or process at the right time, automatically and reliably. Yet what seems simple to describe is difficult to achieve in practice. Lets first look at each of these components of integration in turn.

The Right Data

Data from automatic equipment or from human beings is usually collected for a specific purpose. Since a business enterprise shares a number of common goals, data that may be exchanged between activities within that business will have some commonality of meaning. Understanding which data is the correct data and how to get at it requires an understanding of the application which manages or owns the data, and the syntax of data access allowed by that application.

In addition, for it to be the "Right Data" it must be accurate. Data which is inaccurate leads to false conclusions if used in analysis. It can lead to unhappy customers if it regards the customer's order or availability of product to fill the customer's order. The wrong problem may be attacked if the information is used in process control.

Two of the banes of accuracy in information are human data origination, and human interfaces between software applications. An axiom of accuracy in handling data in computerized systems is that any necessary item of data should be entered once and only once by a human being, or collected automatically from a machine, after which it should be transferred electronically rather than manually.

The Right Format

Data is used for differing purposes within the commonality of goals, and may need differing forms of representation. It is unlikely that one exact format will support the multiple views necessary for multiple uses. Therefore, different applications often have different formats for internally representing data.

An activity needs to access data in its own format, for that data to be useful. There are many dimensions to this requirement. In the simplest dimension, it needs fields of specific data items to be in the appropriate format. If it needs a 16 bit integer, it cannot generally use a 32 bit integer, or a real, or an ascii value. But fields generally exist strung together in records where not only the representation of the field matters, but also its position in the record. Finally, records of one type may be gathered in a file, and the file may contain records of other types, if only as headers or trailers describing the rest of the file.

The Right User

Its critical that the right data flow to the right user. This means first that there must exist a possible path through which the data may flow. Initially it may be "frisbee-net" or "sneaker-net", in other words, hand carried in an off-line storage format. Local area networks (LANs) are now used to allow the automation of this transportation task. Yet even with the appropriate infrastructure, work needs to be done to make sure that the right data gets to the right user, and not to the wrong user.

Users, or processes, are not static and may move from one computer to another reflecting changes in the organization structure or even growth due to improving business -- which is usually the goal of implementing a software application. Yet, if the mechanisms used to move data from one user/process to the next cannot easily be modified to facilitate the movement of the users/processes from one computer to the next, then if will be difficult to get data to the right user.

The Right Time

Information must flow between activities within the useful timeframe dictated by the requirements of the activities. If information is unavailable when needed it becomes useless. Thus, if production schedules need to be calculated, but information regarding the current status of production and of equipment is unavailable, then the calculated schedule will not match the reality of the shop floor. Automating the flow of information between processes can help get data to the right place at the right time.

Automatically

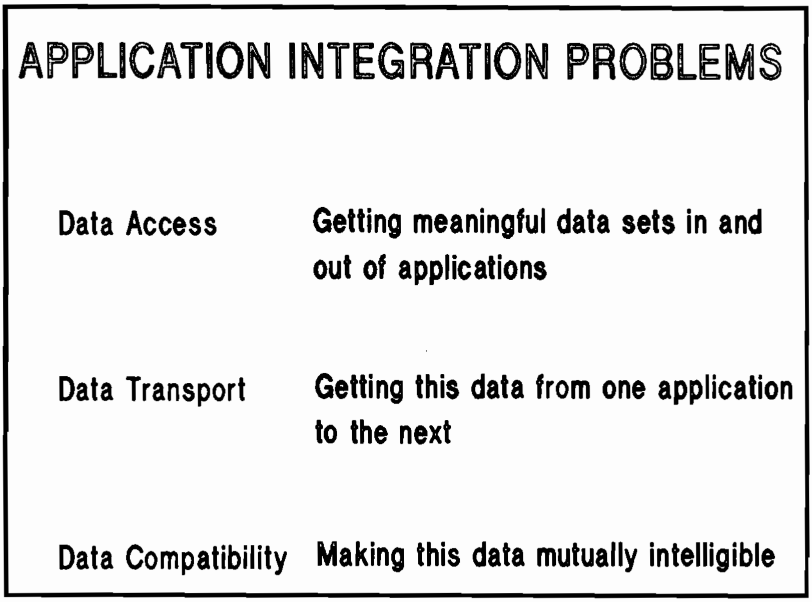
Data that is to flow from one activity to another should be there when it is needed. This is especially true if the time required to gather the data is itself time consuming. Thus, if I need to recalculate a schedule, and this activity can't occur until I have gotten all of the needed data to the scheduler's workstation, then I actually am faced with the task of accumulating the needed data, when what I really want to do is to calculate a schedule. If calculating a schedule is a task that is done frequently, then the required information should be available to my scheduling workstation without my asking so that I can immediately solve my problem, that is, calculate the schedule, rather than expend energies preparing to solve my problem.

Automatic transfer of data can positively impact the accuracy of the data being transfered, if it reduces the number of times the data is directly manipulated by a human being.

Reliably

Automatic access, transfer, and reformatting of data requires that the integration routines used to achieve this purpose function with reliability. A system that operates haphazardly will not be trusted with mission critical funtionality. At a minimum, to behave reliably, an integrated system allowing data transfers between applications or users needs to:

* guarantee delivery of messages
* save and restore messages upon shut-down and start-up
* spool messages if network is down or receiver otherwise unavailable



APPLICATION INTEGRATION PROBLEMS

Data Access — Getting meaningful data sets in and out of applications

Data Transport — Getting this data from one application to the next

Data Compatibility — Making this data mutually intelligible

To integrate software applications, three sets of problems need to be solved. First, meaningful data needs to be accessed from both the sending and the receiving application. Second, this data needs to be transported from the sender to the receiver. And third, the data needs to be made compatible, that is, mutually intelligible, between the sender and the receiver.

For example, moving Bill-Of-Materials (BOM) information from a CAD system to an MRP system requires at a minimum the following steps:

```
* Get the BOM data out of the CAD system     (Data Access)
* Move this data from the CAD computer through the
  network to the MRP computer             (Data Transport)
* Reformat the data from the CAD system's BOM format to
  the MRP system's BOM format         (Data Compatibility)
* Get the BOM data into the MRP system       (Data Access)
```

Data Access Problems

Lets look more closely at data access problems. As shown here, there are a variety of causes of these problems.

Application specific problems include:

- * Matching the different security schemes
- * Differing access syntax (due to application design)
- * Lack of IMPORT/EXPORT mechanisms used
- * No Data Base Management System (DBMS) used

There is a large amount of arbitrary choice in how various functions can be implemented and there may be no systematic solution to matching syntax between sender and receiver.

Additional problems come from heterogeneous DBMS's used which have differing access syntax and different security schemes. At the DBMS level lies the common problem of implicit structure with no documentation of the rules governing data model consistency. Discovering the implicit relationships that exist in the applications, and how violating those relationships causes data integrity problems, can be an adventure when interfacing applications.

The file systems used add a level of diversity in a heterogeneous environment, and require either specific (and expensive) combinations of expertise or a long (and expensive) learning curve to code around the mismatched functionality. Variations are common in naming conventions, file types, file domains, and file access methods.

Finally, integrating applications creates a set of problems involving ownership of data, keeping data both consistent and concurrent between sets of integrated applications, and maintaining data integrity when the integration routines themselves can be a trojan horse, by-passing normal application integrity checking measures.

<u>Data Transport Problems</u>

The biggest problem caused by diversity in operating system and network services is the experience or the learning curve required to employ them. If you have to hire someone with specific expertise, say on both MPE-XL and HP UX, that person will be expensive. The alternative is to train someone on both systems, which also is not only expensive but time consuming.

Additionally, today no single method exists that can perform the functions of:

- *File transfer between local applications
- *File transfer between remote applications
- *Message transfer between local applications
- *Message transfer between remote applications
- *Program control (start/stop) locally
- *Program control remotely

This usually means that the integration routines created to integrate local applications (on the same CPU) need to be completely rewritten to integrate remote applications (different CPUs), thus creating inflexibility by adding both cost and lead time to the movement of applications from one CPU to the next if business growth or organization change dictate that movement.

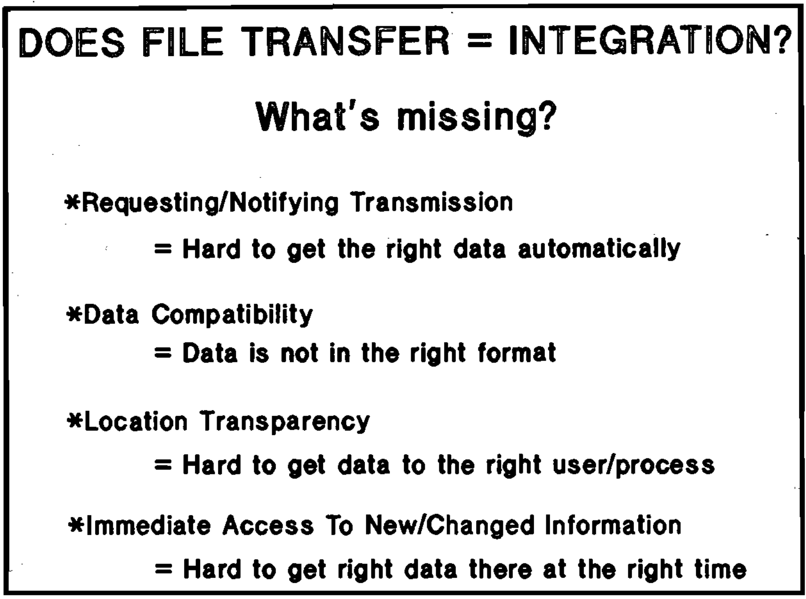
If integration routines create a path for data to flow between applications, it is then also necessary to address the problems of synchronizing data, of sequencing input strings, and of recovery procedures to employ if data transmission fails leaving an incomplete update.

Data Compatibility Problems

There is a hierarchy of problems in making data compatible between applications. At the hardware level, 16-bit computers express the same data differently than 32-bit computers. The DBMS or other subsystem software used to construct applications may also keep data in different formats. Thus in all IMAGE databases, data items must have an even length in bytes. Odd length data items get a byte padded. In SQL databases this isn't the case. Thus moving data from an SQL to an IMAGE database will require padding on any odd length fields.

Programming languages will also differ in how they represent data. TRUE and FALSE may be even and odd in some languages, and zero and negative one in others. Some languages keep two dimensional data in row major order, others in column major order. And so forth.

These differences are systematic. What isn't systematic is the application specific view of data. Thus assume two application products both have a field called ORDER-STATUS, and that this means the same in both applications. Further assume the field is defined in both applications as being one byte long of ASCII character. Even assume the same value "C" is legal in both applications. You still have the case of the same value "C" meaning "COMPLETE" in one application "CANCELLED" in another.



DOES FILE TRANSFER = INTEGRATION?

What's missing?

*Requesting/Notifying Transmission

= Hard to get the right data automatically

*Data Compatibility

= Data is not in the right format

*Location Transparency

= Hard to get data to the right user/process

*Immediate Access To New/Changed Information

= Hard to get right data there at the right time

File transfer technology has existed longer than any other integration technology, and is often postulated as being sufficient to integrate software applications. Most modern application software products also contain IMPORT/EXPORT functions to utilize file transfer. While file transfer is necessary in integration, it is not sufficient since it lacks several critical features that are mandatory to provide the "right data to the right user at the right time in the right format, automatically".

Request/Notification

Files can either be pushed or pulled across a network depending on whether the sender or the receiver initiates the transfer. File transfer schemes generally don't contain request and notification capabilities that allow requesting the transfer of the file, or even the creation of the file to transfer, or notification that the file has arrived. It is not enough to go and get a file from another machine. That file must contain the appropriate data; today's data not yesterday's data, for example.

To make sure it gets the right data, either the receiver must request the sender to build the file and send it, or the sender must build the file, send it, then notify the receiver of its presence. Either case requires some form of messaging. In short, without request or notification, its hard make sure that you're getting the "right data" and its hard to automate that process.

Data Compatibility

The file must contain data in a format that the receiver can understand. This means that representational differences must be reconciled, not to mention the arbitrary differences in application specific semantics. Since file transfer schemes do not usually translate binary data between alternate representation schemes, most application IMPORT/EXPORT functions require the file to be ASCII or EBCDIC. Yet this still forces either the sender or the receiver to understand the other's format and map the data thereto or therefrom. Therefor, file transfer mechanisms cannot get you the data in the "right format".

Location Transparency

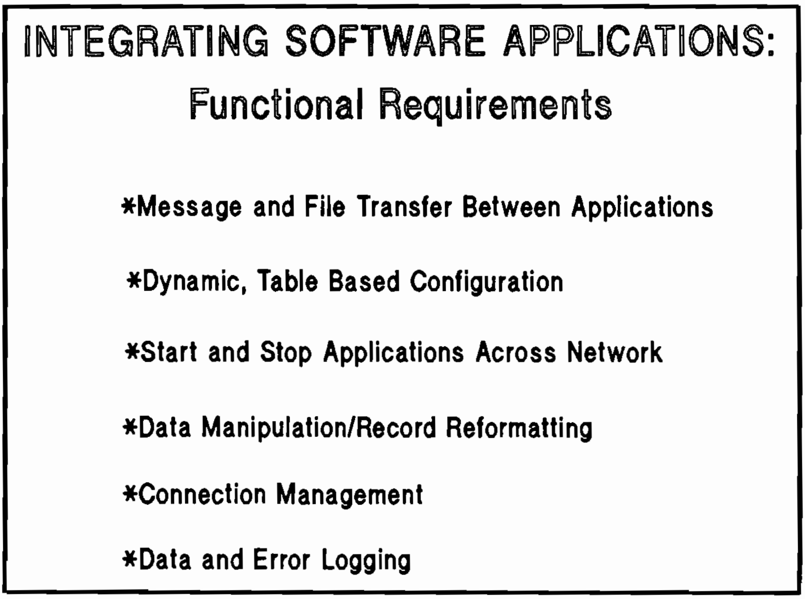
Where a sophisticated system of integrated software is desired, there are many exchanges of data between components. None of these exchanges should have specific information regarding the physical topology of the network. Otherwise, any change to the network precipitates many modifications to the exchange routines. File transfer schemes do not provide location transparency. This must be added on top of the scheme through some form of table lookup and administration of the address tables. Without location transparency, its hard to get data to the "right user".

Batch Nature

But perhaps the biggest shortcoming of file transfer as an integration technology is its batch nature. To keep the various components of an integrated system synchronized requires the exchange of information from its source of origination to every activity that requires it as soon as the information is developed. Thus, as soon as an order is received into a Customer Order Entry system, that order information should be sent to all of the other applications that need it. This requires the capability to receive information whenever its available.

Yet most IMPORT/EXPORT schemes are not constructed to be ever ready to receive messages. They are oriented around the file transfer mechanisms available. This implies a periodic capability, for example once a day, rather than online availability. It is often the case, that the IMPORT/EXPORT mechanisms must be manually initiated, and lack even the syntactical capability to be automated.

Performance of a task often requires the availability of multiple files from multiple sources. The necessity to drive the transfer of those files, and the serial, batch nature of file transfer mechanisms, may well add sufficient lead time to the task to make its performance not possible within the time required, or at best unpredictable.



INTEGRATING SOFTWARE APPLICATIONS:
Functional Requirements

*Message and File Transfer Between Applications

*Dynamic, Table Based Configuration

*Start and Stop Applications Across Network

*Data Manipulation/Record Reformatting

*Connection Management

*Data and Error Logging

Having both defined integration and discussed the problems that make it difficult to achieve, lets look at the functions that are both necessary and sufficient for software integration in a heterogeneous LAN environment.

Message and File Transfer

In order to communicate between products, the ability to send both messages and files is necessary. (Messaging between applications means sending a stream of data from one application to another without passing through a disc-file en route.) Most software applications currently support IMPORT/EXPORT through ASCII files, and this is necessary for systems requiring batch updates. But we have seen how messaging is necessary for synchronization of processes and for notification of the availability of interface files, and for other purposes.

Dynamic, Table Based Configuration

In order to promote a flexible computing environment, it should be possible to rearrange computer hardware without altering application interface software source code. This can be done by associating ID-names to network addresses, using the names in interfaces rather than the addresses, then matching the names to real addresses at transfer time.

Changing the network layout only requires changing the associations of names and addresses. No altering of interface programs is required.

Start and Stop Applications Across Network

Many automated manufacturing processes may have software applications running on systems without operator consoles. Starting and stopping the software application needs to happen in response to other activities that have occurred on the same production line. For example, when a printed-circuit board leaves one insertion machine for another, the application driving the second insertion machine needs to be activated. Other circumstances may require all cell-control type applications to be halted due to emergency situations. Or applications sending data may need to force applications receiving data into activation in order to insure that data transfer succeeds. All of these cases necessitate application control across the network.

Data Manipulation/Record Reformatting

In order to make data compatible between applications, it is necessary to provide data manipulations and record reformatting. This allows isolation of one application from information about how the other application represents data. This isolation makes it possible to change one application (with respect to how it represents data) without forcing the same change to be made to other applications with which it is interfaced. This isolation is a major component of flexibility.

Additionally, much of the code written in a custom-coded interface consists of code used to manipulate data. With data manipulation capability, interfaces could be created with fewer lines of code being written, and thus faster and cheaper.

Connection Management

In order to exchange information between two processes across a LAN, a number of specific activities have to take place. These activities can be called "connection management." The following algorythm for communication shows only a few of the requirements (and only from the sending process's perspective):

- Open communications to receiving computer
- Create session on receiving computer
- Execute receiver program for data transfer
- Execute send routine
- Sender and receiver programs shake hands
- Data is exchanged
- Exchange is verified
- Sender program is halted

- Receiver routine is halted
- Log off session on receiving computer
- Close communications to receiving computer

Clearly any high performance scheme to communicate useful data between applications on different computers does not execute all eleven steps for every communication, but attempts to maintain the sending and receiving programs up and running all the time. If these sending and receiving programs are to be up and running all the time, it makes sense to have all of the sending applications on one machine share the same mechanism, and likewise the receivers.

Data and Error Logging

In a distributed environment, and where many computers may have no operator consoles, it is often necessary to get error information to the system that is being manned. Additionally, data logging makes possible debugging and troubleshooting of applications and interfaces in a networked environment.

Infrastucture Requirements

Certain hardware and software is required to support the functionality described earlier. A Local Area Network, or LAN, supports immediate, high performance connections between computers so that messaging can occur. In addition, network services are needed for inter-process communication and file transfer across the network.

To provide location transparency for processes communicating with each other, some form of message manager or queue manager to handle multiple requests from multiple senders who share one sending process, and to route messages to the appropriate process (And likewise for the receiving process.)

Also required is some form of communication program to execute the sends and the receives. For in order to ensure reliability of message and file transfer, the communication program must handle abnormal conditions such as spooling messages when the receiver is not available, or when the network is down.

While networking hardware and software is currently available that supports both MPE-XL and HP-UX, from Hewlett-Packard and from third parties, the connection management software has not been available until recently. To get this functionality, one had to write the code. This presented the choice of providing identical software on both MPE-XL and HP-UX platforms, requiring portability between the environments, or providing and maintaining separate but interoperable solutions on each platform.

Specific Problems Between HP-UX and MPE-XL

In order to provide integration between applications that are running on HP-UX and applications on MPE-XL , we have determined the functions that are required, and we have looked at the infrastructure requirements. Lets look at some specific problems with providing similar functionality on two different platforms. These problems can be grouped into differences in operating system functionality between the platforms, file system differences, and network services differences. Lets look at each in turn.

Different O/S Functionality

To provide, for example, the message manager and the communication program described above, it is necessary to use some mechanism to hold queues of data in memory, rather than on disk, and to have those queues be able to be shared between processes. On HP-UX, this might be implemented using message queues or shared memory. On MPE-XL this would need to be implemented using IPC or virtual memory.

A number of other HP-UX functions have no direct counterpart on MPE-XL, yet would be useful in developing integration software. These include signals, timeouts using SIGALRM and alarm(), detection of process death, counting semaphores, binary semaphores, fork() and exec(), switching users on the fly, and changing access permissions on open files.

Other functions have counterparts between HP-UX and MPE-XL, but with significant differences which impact portability and interoperability. These include Native Language Support (NLS) functionality, message catalogs, writing to system console, system clock, pipes, and process identification.

File System Differences

File names are different between the two systems and this will impact both the portability of code and the software development process. In addition, access permissions are not comparable between HP-UX and MPE-XL since they both have totally different access schemes. Operations that can be performed on files are also different.

Network Software Differences

Finally, the behavior of networking software tends to be different between the platforms. For example, dscopy and NetIPC have different options, file designators, login information, and general syntax. The ipcselect() function doesn't exist at all on MPE-XL, and cannot be directly emulated.

An Application Integration Solution

Recognizing the difficulties in providing complicated functionality in an identical and interoperating format on both HP-UX and MPE-XL, Hewlett-Packard has created the HP Software Integration Sockets (HP Sockets) product. HP Sockets is a tool to integrate new and existing software applications with minimal or no changes to the applications. It provides functionality specifically designed to integrate software applications:

- Table based communications; both file and message
- Data translation, manipulation, and record reformatting
- Local/remote program control: Start, Stop, Signals
- Simple but powerful Application Program Interface (API)
- Consistent integration method across multiple platforms
- Administration of integrated system
- Simulation of application interfaces
- Data and error logging

HP Sockets is available August 1, 1990 for the HP-UX platforms and is being ported to MPE-XL, with availability expected in 1991.

Table based communications between applications

This feature allows flexibility in configuration, since interface programs can be constructed that do not need source code changes when any of the following interface characteristics are changed:

- Network address of sending or receiving application
- Type of CPU on which the other application resides
- Communication buffer sizes (impacting performance)
- Invocation and execution parameters for process control

Functionality provided includes connection/session management, time-out handling, message spooling, message sequencing, message tagging, guaranteed delivery, incoming message and file notification, synchronous/asynchronous messaging, destructive/non-destructive message reads, mail box, and message access by tags or range of tags. All of this capability is needed to provide reliable data exchange between networked applications without writing a lot of code.

Data translation, manipulation, and record reformatting

This feature greatly reduces the amount of lines of code needed to create an interface between applications, by providing a configuration language that specifies record formats both to the sender and the receiver. HP Sockets then maps the message from sender to receiver's format en route.

Each interfaced program is thus insulated from detailed information about the internal representations used by the other programs with which it communicates. Changes to one program do not necessitate changes to other inter-communicating programs, only to HP Sockets data definition and manipulation tables.

Local/Remote program control

This advanced feature allows the controlling of processes registered to HP Sockets and can be used to automate interfaces by forcing sending or receiving programs into operation, or for task sequencing in constructing production lines by integrating process control software. Both starting and stopping can be limited to specific processes for security purposes.

Application Program Interface (API)

This allows the creation of sophisticated interface routines with minimal programming. The API contains a dozen powerful procedures to send files and messages, receive messages, and start and stop other programs. The interfaces created do not need to establish sessions or connections, manage the communication queues, handshake, spool, or recover from errors.

Consistent across both platforms

The learning curve in integrating applications is reduced because there is only a small number of routines in the API, those routines implement advanced and powerful functionality and by having the same API on both platforms. And since access to the network specific procedures is performed inside HP Sockets, applications can be integrated without regard to the syntax and operation of Network Services (especially DSCOPY and NetIPC).

Administration of integrated system

Administration features include starting and stopping the entire integrated system of networked applications, or selectively starting or stopping individual computers as necessary for flexibility in the operating environment. Also, alternate configurations are supported to allow easy switching between configurations either for hot-backups or for developing interfaces at one site to deliver to other locations.

Shut down occurs gracefully, storing messages that have not yet been delivered; upon start up, these messages may be deleted, delivered, or selectively deleted by criticality. All of this functionality is designed to ease the ability to administer a integrated system of networked software applications.

Simulation of interfaces: Data and error logging

Diagnosis and troubleshooting either while developing interfaces or while the integrated system is in operation is aided by the interface simulation capability, which also provides interactive access to the API routines, and by data and error logging. The error logging can be used also for user errors, to minimize the number of logfiles that need to be checked in troubleshooting.

Conclusion

Integration has been defined and explained as the ability to get the right data in the right format to the right user at the right time, automatically and reliably. We have seen how problems of data access, data transport and data compatibility make this goal which is simple to express, difficult to achieve in practice.

The functions necessary to allow integration were introduced, and some of the difficulties associated with creating this functionality on both the MPE-XL and HP UX platforms were discussed. To alleviate these difficulties, and improve the ability to integrate software applications running on the HP UX platforms with software running on the MPE-XL platforms, Hewlett-Packard has created the HP Software Integration Sockets product, which was briefly described.

Unix Networking Integrated in a Distributed Commercial Environment

David J. Giles
Hewlett-Packard Company
675 Brooksedge Blvd.
Westerville, OH. 43081

Abstract

Unix machines are increasingly becoming more pervasive throughout corporations. Engineering departments have traditionally used Unix in their work, and Unix is starting to appear around corporations to support commercial applications. This, along with some soon to be released products for the HP3000 provides us with the opportunity and the foundation to build distributed, cooperative computing environments. The newness of Unix in commercial environments invariably raises questions as to: how can I connect to these machines and what can I do with them once they're connected? Unix networking brings some innovative products and capabilities to commercial environments; new capabilities seem to be appearing almost monthly. Because of this rate of change, it's becoming important to understand this new technology and formulate a Technology Plan. This paper will introduce you to some of this new technology that can provide you with a foundation for a future distributed computing environment. Topics include: standards-based networking, because it can integrate multi- vendor systems; Network File System; Task Broker; Network Computer System; Application Programming Interfaces; and Graphical User Interfaces.

Why Distributed Computing?

Distributed/Cooperative Computing is expected to be the wave of the '90's, and the power of desktop computers seems to be the nucleus of this concept. Certain computers are good a certain tasks, and distributed computing is designed to exploit those capabilities. For instance, desktop computers a good for data entry because they offer a strong user interface and quick response. Mini computers are good for file storage and transaction serving because they are fast, support large databases, and provide a central point of data archiving. Mini's are also good at manipulating data because they have good retrieval and summarization tools. Desktop computers are good for the presentation and analysis of that data, again because of the strong user interface and good graphics capability. Distributed computing is a good way of harnessing that desktop power and exploiting the capabilities and strengths of individual computers.

The First Step - Getting Machines to Communicate

Although many products exist to connect computers, the goal of this paper is to keep it as multi-vendor and open as possible. Most of the products, except where noted, are widely implemented and available on most vendors hardware.

The first step in integrating Unix and non-Unix machines into business networks is networking. Industry analysts, as well as Hewlett-Packard, predict that client/server computing model, or the synthesis of the centralized mainframe with dispersed PC-based model, as almost universal in the 1990's. The necessary backbone for client/server computing systems is the network.

Hewlett-Packard's vision of future computing, in a nutshell, encompasses diverse computer environments behaving homogeneously to the end user. It's almost by definition that proprietary networking models will not integrate diverse computing environments. Therefore, standard networking will become the key to integrating diverse computing environments, which is a fact of life for many corporations, into the client/server model. Standard networking minimizes the incompatibilities and complexities in connecting diverse computers, but it also gives us greater flexibility to operate in multi-vendor environments. A large Columbus, OH based corporation helps illustrate their computing strategy with the following picture.

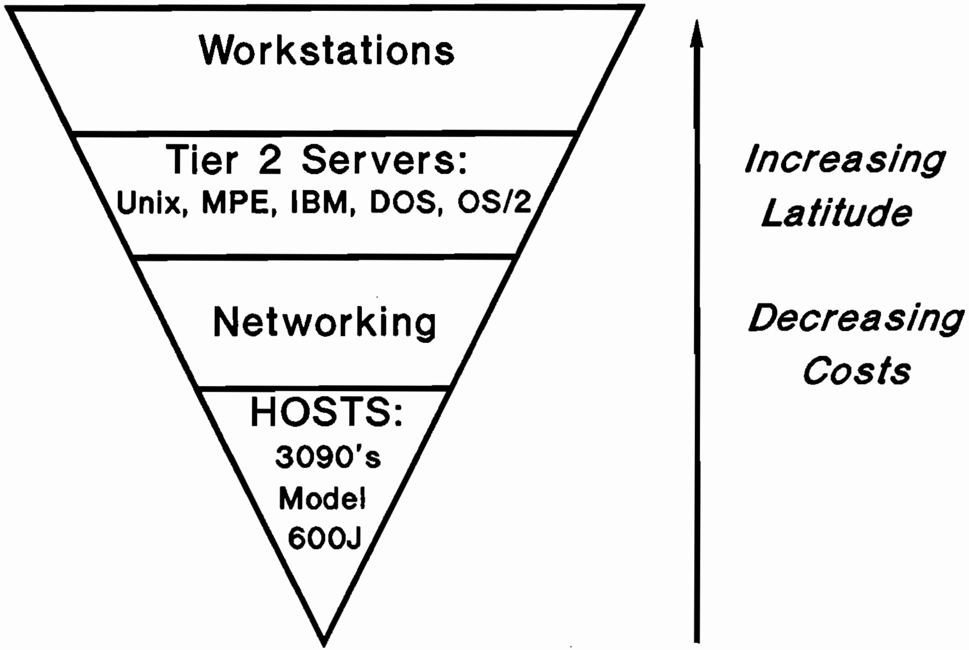
The elements in the lower layers should not change and remain as static as possible, because it's very expensive to change at these layers. As one moves higher up the layers of the triangle they permit users increasing latitude in choosing computing equipment. Although its a simple picture, it illustrates an important point - standard networking forms a foundation for their entire computing strategy.

So what can we do to integrate various machines into integrated business networks?

ARPA/Berkeley and HP's Network Services

ARPA Services are a de facto networking standard in Unix environments, and many non-Unix operating systems also support ARPA Services. Because ARPA is a standard and has multi-vendor support, it provides us with a good tool to integrate various machines into a network. ARPA provides services such as: FTP, which enables file transfers between machines; TELNET, the facility for terminal emulation and virtual terminal support; and SMTP, the Simple Mail Transfer Protocol. SMTP is a mechanism that enables electronic mail to be delivered between machines.

Berkeley Services are also a de facto networking standard in Unix environments, and has the same multi-vendor support, but they are not widely implemented on non-Unix machines as ARPA Services are. For example, ARPA/Berkeley Services are both widely supported on Unix machines, but only ARPA Services are supported on a DEC VAX VMS operating system, and ARPA Services are supported on MPE and in the future, MPE/XL. Berkeley Services are similar to ARPA Services, but Berkeley Services offer more functionality. Both services support file transfer and virtual terminal, but Berkeley additionally supports remote command execution, and "sockets", which is an Interprocess Communication (IPC) mechanism used to write distributed applications. See IPC and API's more information. Berkeley Services also provides a mail facility called Sendmail. It uses the lower-level ARPA service SMTP. It also provides electronic mail across local and wide area networks.



Workstations

Tier 2 Servers:
Unix, MPE, IBM, DOS, OS/2

Networking

HOSTS:
3090's
Model
600J

Increasing
Latitude

Decreasing
Costs

Elements at Lower Levels Should be Static

So what level of network integration can we have with ARPA/Berkeley Services? Both ARPA and Berkeley Services enable us to do file transfers between machines, however Berkeley's file transfer allows us to do more sophisticated file transfers, a remote-to-remote file transfer. See Figure. Since ARPA/Berkeley Services are de facto standards and supported by many vendors, we can communicate in a multi-vendor environment. Large electronic mail networks can be built with the SMTP and Sendmail facilities. The "sockets" mechanism in Berkeley Services allow us to develop distributed applications that can communicate across multi-vendor machines that support Berkeley Services.

ARPA and Berkeley Services are frequently referred to as a single product as the term ARPA/Berkeley suggests. This can be somewhat misleading. ARPA Services and Berkeley Services provide similar services, but because a vendor supports ARPA does not imply that Berkeley is supported, or vice-versa. Many vendors support both, but there are a few that don't. One example is the HP3000. ARPA features have been announced, but not Berkeley features.

Network Services, or NS3000/XL and NS9000, is Hewlett-Packard's proprietary networking software that enables HP3000's, HP9000's, and other selected vendors computers to communicate. Network Services allows us to connect HP3000 - HP3000, HP9000 - HP9000, and HP3000 - HP9000. When connecting HP9000's using NS, NS enables us to do Network File Transfer (NFT), Remote File Access (RFA), and Resource Sharing of discs and printers between HP9000's. When connecting HP3000's to HP9000's with NS, we are able to do NFT between any two machines on the network, and do Virtual Terminal (VT) to the HP3000, but we can't VT from the HP3000 to the HP9000 - it's one-way only.

Here is an illustration of how one company users NS to provide a service for their design engineering group. A Columbus, OH manufacturing company implemented NS between their HP3000 business system, and their HP9000 design workstations over an IEEE 802.3 Local Area Network. They use the HP3000's large disc capacity to store product designs and drawings, and to ensure regular and consistent archival procedures. Since HP-UX filenames are longer than MPE allows, an HP-UX shell script manages a database of HP-UX to MPE filename mappings. The script automatically sends and receives drawings from the HP3000 using the NS DSCOPY command. ARPA/Berkeley Services could have provided this same functionality, plus some, but ARPA was not supported on the HP3000 at the time.

The NS Virtual Terminal, VT, capability to the HP3000 can be a really useful feature, particularly if your are running X-Windows on an HP workstation. The HP-UX 7.0 release includes a program, VT3K, which opens an MPE window on your workstation, enabling you to run HP3000 block mode applications within a window on your workstation. This can be very useful to an engineer designing a part on a workstation, for example, because he can simultaneously view the part he is designing, and view the manufacturing Bill-of-Material that's stored on the HP3000.

NS also allows limited multi-vendor connectivity - only to a DEC VAX. The only service supported is file transfer.

Virtually all features contained in NS are contained in ARPA/Berkeley Services, but not all features contained in ARPA/Berkeley exist in NS. The primary difference between NS and ARPA/Berkeley is that ARPA/Berkeley is a multi-vendor standard, offering multi-vendor connectivity, whereas NS is primarily for HP computers only. Both ARPA/Berkeley and NS can co-exist on the system, so you can have the benefits offered by both services.

The networking products just discussed give us three basic functions: file transfer, terminal emulation, and mail. Generally, none of these functions demand high-performance networks. Terminal emulation just has to keep up with the end user, which is measured in seconds rather than milli-seconds, simple file transfers are done fairly quickly, and who really expects mail to be delivered instantaneously anyway? So network performance really hasn't been much of an issue. These next products however, essentially transform the network into the computer. Rather than upgrading to a larger, faster, and a more expensive computer to gain horsepower, we can now simply "plug-in" new computers into the network and utilize their CPU power immediately. Just as SPU performance management is an important part of a System Managers job today, so will network performance management in the future.

The Next Step: Getting Machines to Work Together

The next level of integration, once machines are able to communicate, is getting them to work together in a more cooperative environment. Again, without a well thought-out network strategy, it will be harder and more expensive to achieve this. The following products enable tighter integration of heterogeneous computers to bring real benefits to end users.

The Network File System

The Network File System, NFS, is a mechanism which enables file systems on remote computers to "attach" themselves to file systems on local computers. To the end user this is all transparent because the remote file system, once mounted, appears to be a local file system. Files residing on the remote file system are transparently accessed just as local files are.

The Network File System is an ideal tool to distribute data across a network. A typical installation using NFS could have a file server, usually a powerful minicomputer supporting a disc-farm, with workstations connected to the minicomputer server through an IEEE 802.3/Ethernet network.

Virtually all Unix vendors support NFS on their systems, and some vendors support it on their non-Unix systems. So it's an ideal way to distribute data across a network.

NFS is an excellent tool for building distributed applications and fits well into the client/server strategy. Now lets advance a level of sophistication in integration to more "intellegent" networks. One industry analyst describes client/server systems capability in the following way: "A client system should decide whether it is capable of performing the server task, and if not, pass it to an appropriate server."

Task Broker

Task Broker is an application developed and used internally at Hewlett-Packard which allows a client system to decide which computer on the network is most capable of servicing a task, and passing it to the most appropriate server. When a user on a client machine submits a task or job to Task Broker, Task Broker queries each machine on the network about its ability, or affinity, to service this task. The client machine collects and compares the bids from each of the servers on the network, and the client machine, by choosing the highest bid, submits the job to the most appropriate computing resource on the network.

Client machines are also free to bid on their own tasks, so it's possible, if there are no available network computing resources, that a task will be run on a client machine without being distributed at all.

An important concept in Client/Server computing is that tasks are passed to the "most appropriate computer." Task Broker chooses the most appropriate computer by comparing "affinity values", or bids, returned from the computing servers. The affinity value is calculated through user customizable shell scripts which gives the System Administrator great flexibility to create complex affinity programs. In its default configuration, a server bidding on a task will return an affinity value based on its average machine load calculated over the past few minutes. Affinity values range from 0 to 999; an affinity value of 0 means that a server is not able to service this task. A high affinity value means that this server is well suited to service this task.

Task Broker allows computers to bid on specific classes of services like "simulations", or "compiles", enabling a specific computer to service specific "classes" of requests. For example, a minisupercomputer may bid highly on a CPU intensive task in the "simulation" class, whereas a specific minicomputer may bid highly on a task in the "compile" class.

A good example of a Task Broker application is a large compile job, one that may take many hours to run on any single computer. A Unix programmer could write a "make" script that invokes Task Broker to parcel out the program compiles to computers across the network. The script then would wait for all compiles to complete before linking them together into an executable program(s). On a single machine this could take hours to complete, but with Task Broker distributing it across the network, it could complete in minutes.

Task Broker requires no programmer support to implement, but rather, it was designed to be implemented by users and Systems Administrators.

The Network Computing System

The Task Broker distributes a single application to a single server on the network based on the servers workload level and capability. It doesn't require programmer effort to implement. Advancing to a higher level of distributed computing and a more sophisticated model of cooperative computing is a product named the Network Computing System, or NCS. NCS is a products comprised of several elements, the Remote Procedure Call, RPC; the Network Interface Defination Language, NIDL; the Network Data Representation, NDR; and the Location Broker. Industry analysts have also characterized the client/server model in this way: "In advanced versions of the client/server model, the server task may be broken into subtasks, which are then assigned to suitable servers."

Developed by Apollo Computer Corp., NCS is a mechanism that enables a program on a client machine to call subroutines that reside and execute on remote server machines. The client and server machines can be multi-vendor equipment.

To implement NCS, a programmer must develop or modify an application into subroutines. The programmer uses the NIDL to define and describe interfaces to the remote procedures. The NIDL adds no executable statements. The NIDL includes a compiler to compile the subroutines into network executable subroutines. An application on a client system makes Remote Procedure Calls to call and execute the subroutines on the server systems. Because the client and server systems can be heterogenous equipment, (e.g. PA-RISC, Motorola, Intel), and each computer will have different internal data representation, the NDR is needed. This facility defines a data-encoding structure between computers to provide a high level of portability and independence. The Location Broker element of NCS "registers" the location of the modules, ie: what machine(s) the subroutines reside on. When the application is run and an RPC is made, the Location Broker is invoked and queries its database for the location of the subroutine and then executes it. Because the Location Broker manages subroutines and their addresses aren't hardcoded into the application, this gives both the programmer and the system administrator flexibility to build dynamic, machine-independent applications.

The difference between Task Broker and NCS is that an application running under Task Broker will only run as fast as the server it's running on, but an NCS application uses the speed and power of many computers on the entire network. One NCS user, Italtel, the Italian telephone company, used NCS to split a computation job across 10 workstations, and cut the run time from 20 hours to just 2.5 hours.

Task Broker and NCS are very complementary products. Although NCS's Location Broker provides transparency by knowing which machine can offer a service, it doesn't provide the load-balancing feature that Task Broker does. Integrating the features of Task Broker into NCS will provide the benefit of a self-balancing "intelligent" network.

The goal of NCS is to provide more CPU horsepower for applications. But most commercial applications aren't compute-intensive, they're I/O-intensive. Commercial applications for NCS will probably be focused on specific applications; one possibility is for financial services applications, and another is for users to inquire into a a central database of information. As people apply creativity and ideas more NCS commercial applications will materialize.

Multi-vendor support for NCS is likely; NCS has been licensed to approximately 200 computer vendors, including HP, IBM, DEC, Cray, and Sun.

The previous products enable you to build a foundation to distribute systems across networks. Graphical User Interfaces are necessary for integration, interoperability, and ease-of-use of the distributed systems.

Graphical User Interfaces

It's been said that "the workstation is the window into the network." The workstation, whether it be a Unix, MS-DOS, or OS/2 workstation, is a key element in Hewlett-Packard's cooperative, or "NewWave", computing vision. It's difficult to have an workstation operating system that's "everything to everybody" but ideally, these diverse operating systems should be transparent to the end users. That is, they should behave consistantly. Otherwise, this diversity creates roadblocks toward natural, easy-to-use cooperative computing. Graphical user interfaces shield the complexities of an operating system from unsophisticated users. On OS/2 machines it's Presentation Manager, and on Unix workstations it's X-Windows/OSF Motif.

X-Windows is the standard user interface in the Unix world. In to being a standard user interface, it also give the users great power in distributing applications across machines supporting X-Windows. Virtually every computer manufacturer supports X-Windows, and it runs on non-Unix operating systems as well. Because X-Windows was designed to be network independent, it can be used over "any reliable byte stream," several of which are standard and available on most computers. X-Windows enables a user to run clients (processes) on any remote computer on the network but have it display on his local workstation server (monitor.) Since the workstation is Unix based, multiple clients can be running on machines throughout the network, and each displayed on his local server, thus letting a user "work" in multiple applications at the same time.

For the last few years, Engineering departments have been exploiting the productivity, ease-of-use, distributed computing, and other benefits of X-Windows. Hewlett-Packard has announced its plans to support X-Windows on MPE/XL in late 1990 thus enabling tighter integration of the capabilities of MPE/XL throughout a multi-vendor distributed network.

OSF/Motif is a graphical interface built on X-Windows, and is recognized as an industry standard. The Open Software Foundation (OSF) chose Motif because it transparently bridges the differences in look and feel of Presentation Manager and X-Windows, so it's easy for a PC user to move to a Unix-Workstation because they will "behave" the same way. For all intents and purposes of this discussion, X-Windows and Motif are equivalent.

Hewlett-Packard has announced plans to port NewWave onto the OSF/Motif environment. The first phase of this plan of full impementation of NewWave, is the Visual User Environment (VUE.) This give users a true graphical user interface, making Unix much easier for end users to use. The VUE enables you to start applications by clicking the mouse on an icon or by calling up a pull-down menu. The VUE is integrated with the file manager so that a data-file icon can be selected, dragged and dropped onto an application for execution. When NewWave is fully integrated more sophisticated capabilities will be availaible. Capabalities such as agents - to automate repetitive tasks, and hot links, which automatically manages compound documents. The features contributed from NewWave, merged with the distributed environment X-Windows/Motif contributes, promises to be a powerful combination. NewWave already runs on MS-DOS, is planned for OS/2 and HP-UX. This will give the end user a common operating environment across all three platforms.

So how might X-Windows/Motif be used in a commercial environment? Suppose your companies sales history is on a VAX, inventory system is on an HP3000, and a product design is on your local workstation. From your local workstation, you can view the product design, the bill of materials, and the sales history for that product, plus be running a spreadsheet on your workstation, simultaneously.

A tight level of application integration is necessary to fully use all of the features of X-Windows/Motif, and of course, this requires C-language programming. And it's not a trivial task either. However, many companies are starting to offer software tools which genenerate the necessary X-Windows/Motif/NewWave code to design some really stunning applications. These tools will make graphical user interface programming simple.

PC-Integration

Personal Computers are important elements in any Client/Server computing strategy. Let's look at ways they too can be integrated into our business network.

Starting at the most basic level is the terminal emulation and file transfer capabilities that packages like AdvanceLink or Reflections give us. Both run over serial or local area network connections.

Advancing up a level of functionality and also standardization is ARPA services for the PC. ARPA services enables your PC to do file transfer, and virtual terminal (TELNET) communications to other computers running ARPA Services. ARPA for the PC also supports some of the Berkeley Services features such as sockets, and remote file transfers. Berkeley sockets is a mechanism that you can use to write distributed applications. More information on sockets, named pipes, and message files is ahead. ARPA services requires that a local area network be in place because it doesn't run over serial connections.

The next level of PC-integration includes PC-NFS, the Network File System for MS-DOS computers. NFS better fits into the client server model than ARPA Services, and it give us much of the same functionality that NFS does that runs on the larger multi-user computers. NFS running on multi-user machines can act in both the client and server capacity, however PC-NFS can only act in the client capacity. Of course, PC-NFS can integrate with any system running NFS, most of which are Unix systems. Just as NFS does, PC-NFS "attaches" physically remote file systems to your local DOS directory, and the end user perceives them as just another directory on his PC. PC-NFS also provides basic functionality of allowing PC's to spool print files on remote NFS printers.

Advancing another level of functionality and integration potential brings us to Lan Manager and Lan Manager/X. Lan Manager promises to be the new standard in network operating systems for MS-DOS and OS/2 PC's. Lan Manager was developed by Microsoft and is backward compatible with MS-NET, Microsoft's first generation network operating system for MS-DOS computers. Lan Manager offers all of the capabilities found in MS-NET, and then some. Major software and database vendors have announced support for Lan Manager and will develop Lan Manager based products.

Lan Manager enables MS-DOS and OS/2 PC's to be servers for PC networks, and larger servers will be available as vendors adapt Lan Manager to their mini and mainframe computers. One such port that is available today on HP-UX is Lan Manager/X (LM/X). LM/X allows HP-UX machines to be servers for PC networks. This provides a wider range of processing power and scalability than current Intel-based PC's allow. As the name obviously implies, Lan Manager and LM/X run over local area networks. Both IEEE 802.3/Ethernet and Token Ring LAN's are supported.

So what sort of functionality and integration can we get from LM/X? LM/X provides users with the ability to create, delete, and modify files on the Unix server system. Users can also configure shared resources such a discs, printers, and plotters, so each PC doesn't have to be outfitted with these expensive peripherals. LM/X includes a robust spooling mechanism, which supports features including: routing to first available printer, routing to a specific printer, and multiple-printer priorities. LM/X provides comprehensive security mechanisms which are designed to cooperate with the built in security mechanisms in HP-UX. LM/X meets the DOD C2 level of security. The combination of Unix security and LM/X security provides user and resource level security. Resource level security allows access to a resource to be protected by a password. User level security allows user access to servers/resources for which he has account and access privileges. LM/X also provides record locking mechanisms which are a combination of Unix locking and DOS locking mechanisms. LM/X also provides tools for the software developer to write distributed and cooperative applications. These are called Application Programming Interfaces, or API's.

Interprocess Communication and API's

A distributed application is one with functionality split between multiple computers. Interprocess Communication/API's are used so a program on one machine can communicate to a program on another machine. It's like in the old navy movies, a Captain screams into a pipe, and (hopefully!) somebody is listening on the other end. API is a big term for a nice interface to IPC facilities. Developing distributed applications with API's does require programming on your part, but since these API's are standard, or an emerging standard, it's possible to buy off-the- shelf "shrink-wrapped" applications that use them. Here is a brief description on some of the major IPC/API's available today.

NetBIOS is an IBM de facto standard for PC to PC communications. Both MS-DOS and OS/2 PC's support NetBIOS, but it is weak for building PC-to-Mini distributed applications because it's hard to do internetting. Internetting is the ability to communicate via routers and gateways to computers on remote networks. This feature is probably necessary if you have large, interconnected networks. IBM is moving toward the APPC (LU 6.2) API for building distributed PC-to-Mini/ Mainframe applications.

Berkeley Sockets is a de facto standard for Unix. This makes it a good mechanism for building distributed applications between Unix machines and other machines that support Berkeley sockets. ARPA Services for the PC supports Berkeley sockets, so you can distribute applications between PC's and Mini's also. Berkeley sockets supports internetting, so it's a good candidate for PC-to-Mini and Mini-to-Mini integration over large networks.

Berkeley Sockets is an interface to TCP/IP, which comprises lower levels in the seven-layer OSI stack. This makes Berkeley Sockets dependent on TCP/IP based networks. The other IPC/API's mentioned are more independent of the lower layers of the OSI stack, and this independence gives you more flexibility in running these IPC/API's over a variety of protocols.

Named Pipes/Mailslots are an integrated part of LM/X. A Named Pipe is a bi-directional facility for process communication, which operate very much like Berkeley Sockets. A Mailslot is similar to a Named Pipe, but one-way communication only. Named Pipes/Mailslots don't currently support internetting; hopefully it will be available shortly. Many vendors such as Novell, and Excellan have also announced support for Named Pipes.

NetIPC is Hewlett-Packard's proprietary IPC/API implementation. Since NetIPC runs on HP9000's, 300's and 800's, HP3000's, MPE and MPE/XL, RTE-A, and MS-DOS, it's a good way to distribute applications across HP machines. It's currently the only IPC facility supported on the HP3000. Regarding Hewlett-Packard's focus on standards, for PC-to-Mini applications, you're probably better served by using Named Pipes/Mailslots, or Sockets.

Relational Databases

No discussion on integrating commercial machines would be complete without mentioning something on databases. Relational databases are the standard, and distributing a database across more than one machine will be common. With Image, this would be hard to do. How would you update the pointers across machines and maintain data integrity? Relational databases make distributed processing easier. Many relational vendors, such as Oracle with SQL*Net, and

Hewlett-Packard with Allbase/Net, provide distributed, location transparent databases today. If you plan to migrate toward a distributed computing architecture, relational databases should be seriously considered.

Conclusions

This paper has attempted to outline some products that can integrate existing Unix and non-Unix machines into an integrated commercial network. Although many strategies exist for tying machines together, this paper focused on networking found in Unix environments because Unix networking offers the broadest potential for multi-vendor connectivity. Once the basic networking was described, products were discussed that allow you to build on the networking foundation for a more distributed and cooperative architecture. Graphical User Interfaces were then introduced to make this distributed architecture more transparent, integrated, and easy to use.

PA-RISC is a registered trademark of Hewlett-Packard Company

Unix is a registered trademark of AT&T

NFS and PC-NFS are trademarks of Sun Microsystems, Inc.

Migrating Factory Applications to HP-UX

By

Kent Garliepp

Industrial Applications Center
Hewlett-Packard Company
1266 Kifer road
Sunnyvale, California 94086

At the present time, many factory automation programs on HP 1000 computers connect directly to the factory floor via communications with RS-232 devices. A number of these factory automation programs communicate to factory floor devices using the PCIF/1000 interface program.

While HP 1000 computers and PCIF/1000 are still a strong presence in the automation arena, some developers are moving to HP-UX platforms in order to use the advanced features available on HP 9000 series computers.

This paper will discuss an application migration strategy to HP-UX computers for applications using PCIF/1000. The strategy will be exemplified by an example conversion of PCTST (PCIF test program) to run on HP-UX using **Hewlett-Packard Device Interface System (HP DIS)** to create a device interface.

HP DIS is a tool kit for creating software interfaces between HP 9000 systems and intelligent factory floor devices that use serial communications. **HP DIS** allows the rapid implementation of specific protocols from a state-transition description of the behavior of the factory floor device.

Introduction

Many industrial computer users are interested in migrating application programs to HP-UX computers to take advantage of the latest technology or features provided in an environment that is rapidly becoming a standard. Much of the conversion consists of getting original source code to compile. Other problems may arise during migration in the areas of user interfaces and data bases to take advantage of the technology available on the HP_UX computers. New user and database interfaces usually require a rewrite of the application code to accommodate them and make use of their advanced features.

Factory control and monitoring application programs have these same migration problems. They also have the additional problem of migrating their factory floor device interfaces, typically interfaces to programmable logic controllers (PLCs).

Factory floor device interfaces may be built into the factory control and monitoring application but are typically a separate device interface application. This application is a stand alone module that provides access by factory control and monitoring applications to multiple types and brands of factory floor devices in a transparent fashion. The interfaces are usually tightly coupled to the features of the computer operating system. An example of this is Programmable Controller Interface/1000 (PCIF/1000) running on HP1000 series computers using the RTE operating system. Since such a device interface is tightly coupled to the features of the operating system, it cannot migrate without a major rewrite and creates the same dilemma as having to rewrite application code to use the features available on the target computer.

This paper will address the problem of providing an interface on HP-UX computers that will allow factory programs using PCIF/1000 on HP1000 computers to migrate without rewriting the application code. This paper will not describe an entire "ready to go" interface to completely replace PCIF/1000 on HP-UX but rather give an insight or strategy to the approach for developing a factory floor device interface that will provide most, if not all the features used by an factory control and monitoring application.

Strategy

The strategy consists of four basic parts:

- Choosing the division of functionality.
- Making assumptions about the functionality.
- Creating a design based on the functionality.
- Implementing the design.

The Division of Functionality

In any design two of the first considerations are the functionality desired and the scope of the design. The designer must decide what the final product will or will not provide, and how they will implement a system that meets these needs. PCIF/1000 is a very complex product attempting to meet all anticipated needs of many unknown projects. In designing it's replacement on an HP-UX computer to satisfy the present and future needs of a specific factory application, the designer can concentrate only on the functionality presently used and design for future expansion needs. In some cases different functionality may be desired because of other changes, such as a new user interface or the incorporation of a new database.

Once the functionality and scope have been determined, the designer must consider the modularity of the design. In this migration problem of an existing application already using PCIF/1000 the designer will need to replace the programmatic interface and the PCIF/1000 subsystem functionality. The problem here is one of partitioning. Since the internal details of the system may not be known this will require much imagineering, that is, reconstruction of a system to provide the same functionality. Figure 1 demonstrates the problem of partitioning. The entire length of the box represents all the functionality of the design of the application through the hardware interface. The movable partitions (dotted lines) separate the application program, the programmatic interface, protocol related functions, and the hardware. In this case the application program is already defined (PCTST) and the hardware interface is already defined as the HP-UX IO drivers (input / output processes) and the RS232 serial MUX hardware provided with the computer. Assuming fixed boundaries of not changing PCTST and using the standard RS232 interface, the only choices for development are the programmatic interface and the protocol related functions. For simplicity, the programmatic interface will implement a minimum amount of the functionality. Most of the functionality will reside in the protocol related functions.

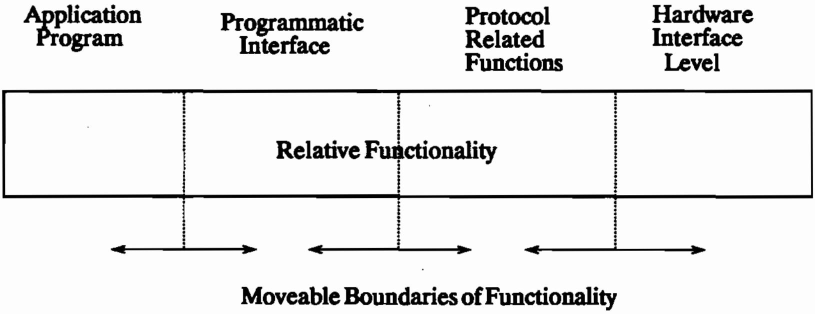


Figure 1. Effort or Complexity

The strategy consists of creating a programmatic interface to the factory control and monitoring application and a finite state machine (FSM) [reference 1] programmable interface to the factory floor devices . While finite state machines can be developed in the C language [reference 2] or with lex and yacc, this paper will use the

Hewlett-Packard Device Interface System (HP DIS) [reference 3, 4] to create a factory floor device (FFD) interface. HP DIS provides a predetermined structure for our example. Using the HP DIS interface, a finite state machine will be developed to provide some of the functions provided by PCIF/1000. The programmatic interface will be developed to replace part of the PCIF/1000 programmatic interface also known as PCARs.

HP DIS provides the capability of defining finite state machines, and it also has many protocol related functions available in action routines. Any actions not already available can be written in C and used in the finite state machine (FSM) as user written action routines. Figure 2 shows schematically a programmatic interface linked with a factory program communicating with a factory floor device interface which implements the protocol related functions. This in turn communicates with the HP-UX IO drivers and hardware and ultimately the factory floor devices. The arrows indicate the data interfaces.

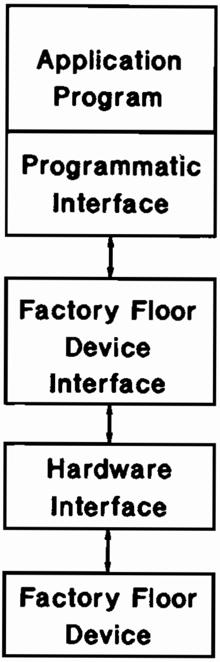


Figure 2. Program to Device Interface

Functionality Assumptions

The migration example in this paper will be based on the following assumptions:

- PCTST will be interfaced with one PLC and run on HP-UX .
- Minimal programming in the programmatic interface is desired.
- Host initiated messages will be synchronous. A new message will not be processed until a reply has been received or a time out occurs.
- Only one type of factory floor device will be used. It will be capable of originating messages.
- Only one port will be accessed by the factory floor device interface.
- HP DIS will manage the data interfaces as FIFO (first in, first out) queues.
- HP DIS manages all IO port processes.

PCIF/1000 Functions

The PCIF/1000 functions that will be shown in the example to demonstrate the strategy are:

PCIF_OPEN	(Connect the application to PCIF.)
PCIF_CLOSE	(Disconnect the application from PCIF.)
PC_PCSTAT	(Get device status.)

HP DIS Functions

The HP DIS functions that will be used in the example are:

DcStartPI	(Start FSM modules.)
DcCleanUp	(Stop FSM modules and remove FIFO queues.)
DcStatPI	(Check if FSM module is running.)
DcSendToQ	(Send a buffer to a FSM's FIFO queue.)
DcRdFromQ	(Read a buffer from a FSM's FIFO queue.)

The Message Formats

The following message formats will be assumed in the model. All capitalized names represent individual ASCII characters and all lower case names represent variables. These message formats are assumed to contain all the data necessary to implement a simple PCIF like interface and represent a PLC device. The data names are defined as follows:

device-#	The logical number of the device.
function-#	A number to represent the function.
key-#	A key for request / response matching.

▪ data-address	A PLC address.
▪ data	A collection of characters (bytes).
▪ status	A transaction result code.
▪ device-to / from	The transmitting and receiving device numbers.
▪ reply-flag	Device initiated flag.
▪ bcc	Block check character.

The structure of the messages are as follows:

- Output message from the programmatic interface.

  device-# function-# key-# data-address data

- Returned message to the programmatic interface.

  device-# status key-# data

- Output message from the host initiated message FMS module.

  DLE STX header data-address data DLE ETX bcc

- Reply message from device.

  DLE STX header data DLE ETX bcc

- Device initiated message.

  DLE STX header data-address data DLE ETX bcc

- Header format.

  device-to device-from function/reply-flag status key-#

Device Interface Model Design

Figure 3 shows a model for host initiated messages. The factory control and monitoring application program sends and receives messages via FIFO queues managed by HP DIS. The message flow to and from the factory floor device is controlled by a factory floor device interface (FMS) module through FIFO queues, a read process, and a write process, all managed by HP DIS . The read process has the capability of recognizing the start and end of a message and rejecting garbled messages or noise.

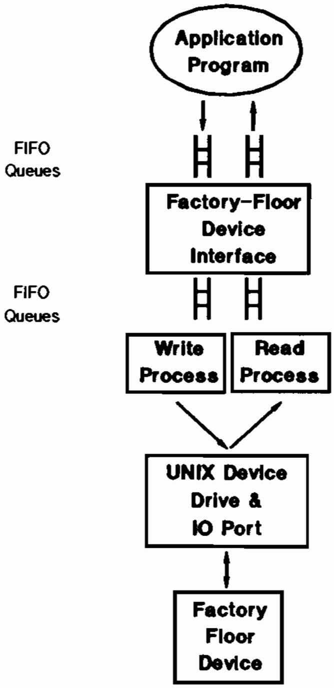


Figure 3. Host Initiated Message Model

Figure 4 shows the model for both host and device initiated messages. The differences between device initiated and host initiated are the addition of the second finite state machine and it's FIFO queues. All messages from the factory floor device are routed to the right hand (device initiated) message module and separated into replies and device initiated messages. The replies are routed to the host initiated module for processing and returned to the factory application program. The device initiated messages are processed and returned to the factory application program as unsolicited messages. The device initiated FSM module will be responsible for handshaking with the device, thus it must also write to the FIFO queue associated with the write process. This is the model that will be demonstrated in implementing the design.

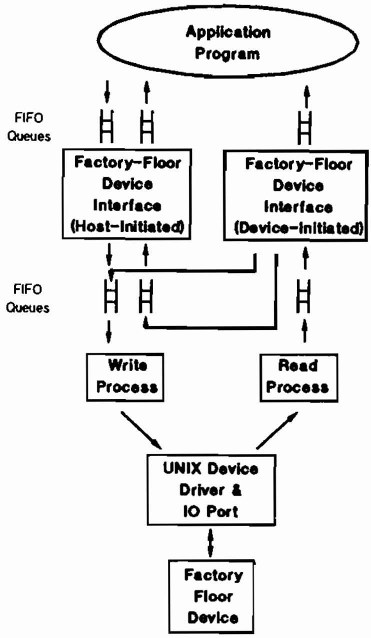

Application Program
FIFO Queues
Factory-Floor Device Interface (Host-Initiated)
Factory-Floor Device Interface (Device-Initiated)
FIFO Queues
Write Process
Read Process
UNIX Device Driver & IO Port
Factory Floor Device

Figure 4. Host and Device Initiated Message Model

The host initiated message FSM module is shown in figure 5. The module waits for a message from the programmatic interface in the IDLE state. Upon receipt of a message, it triggers the send-message event that causes a transition to the WAIT state and executes actions that build an outgoing message and deliver it to the write FIFO queue. The FSM module then either receives a reply or times out. A reply causes a transition to the IDLE state executing actions that send the reply message to the application. A time out causes a transition to the RETRY state to determine if the message should be resent. The unexpected-reply event in the IDLE state removes messages from the FIFO queue that arrive after the time-out (a possible race condition).

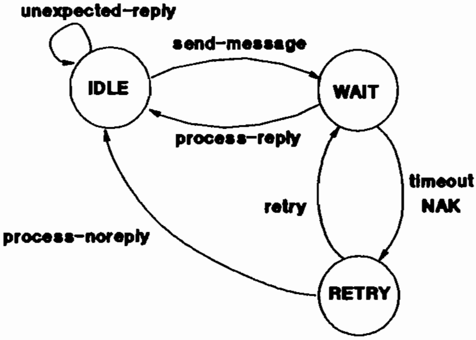


Figure 5. Host Initiated message FMS Module

The device initiated message FSM model is shown in figure 6. The module waits for a message from the factory floor device in the IDLE state. Upon receipt of a message, it triggers the process-message event that causes a transition to the IDENTIFY state and executes actions that will either process the message as a reply or as an unsolicited message.

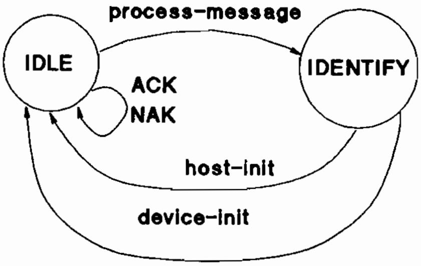


Figure 6. Device Initiated Message FMS Module

Input Matching

The HP DIS read process will manage the reception of entire messages. For example, in the following device message format all the characters are eight bit bytes.

DLE STX chr chr chr . . . . . chr DLE ETX bcc

DLE, STX and ETX are constants and bcc is a variable block check character. The HP DIS read process will search the input buffer for this sequence of characters and write them to the FIFO queue associated with the device initiated message FMS module.

If the message is garbled such that the characters are arranged as shown below, all

DLE STX chr chr chr . . . . . chr DLE STX chr . . .

the characters prior to the second DLE STX will be placed into an unmatched FIFO queue for disposition as an unusable message.

Implementing the Design

This section will show partial code segments and design outlines to demonstrate the coding of the programmatic interface, the coding of the FSM modules and a HP DIS configuration file. The device configuration for the FSM module is based on the model in figure 4.

Programmatic Interface

The programmatic interface will connect PCTST, a program written in Pascal, to the FSM modules via the HP DIS external functions. The programmatic interface is written in Pascal to be compatible with PCTST and implements a minimum set of code to achieve the message formats already described, or to complete simple functions.

```
{ Program fragments for a PASCAL interface for PCTST. }


{ External HP DIS C function declarations start here. }


FUNCTION  DcStartPI ( VAR FSM        : string;
                      VAR configfile : string); EXTERNAL C;
          {EXTERNAL C Function to start up a FSM module}

FUNCTION  DcCleanUp ( VAR configfile : string); EXTERNAL C;
          {EXTERNAL C Function to stop all FSM modules}

FUNCTION  DcStatPI  ( VAR FSM        : string;
                      VAR pid_number : integer); EXTERNAL C;
          {EXTERNAL C Function to get the status of a FSM module}

FUNCTION  DcSendToQ ( VAR FIFO_queue : string;
                      VAR msgbuff    : string;
                          buflen     : integer;
                      VAR Tag        : string); EXTERNAL C;
          {EXTERNAL C Function to write to a FIFO queue}

FUNCTION  DcRdFromQ ( VAR FIFO_queue : string;
                      VAR msgbuff    : string;
                          buflen     : integer;
                      VAR msglen     : integer;
                          waitflag   : integer;
                      VAR Tag        : string); EXTERNAL C;
          {EXTERNAL C Function to read from a FIFO queue}

{ PCIF/1000 equivalent procedures begin here }

PROCEDURE PCIF_OPEN ( VAR stat : ShortInt);

VAR
   dummy      : Integer;
   pid_number1, pid_number2 : Integer;

BEGIN
   { If the interfaces are not running, start them. }
   dummy := DcStatPI ("host_init", pid_number1);
   IF (pid_number1 <= 0) THEN
          dummy := DcStartPI("host_init", "PCIFconfig");
   dummy := DcStatPI ("dev_init", pid_number2);
   IF (pid_number2 <= 0) THEN
          dummy := DcStartPI("dev_init", "PCIFconfig");
   dummy := DcStatPI ("host_init", pid_number1);
   dummy := DcStatPI ("dev_init", pid_number2);
   IF ((pid_number1 > 0) AND (pid_number2 >0))
       THEN stat = 0;
       ELSE stat = 17;
END;
{-----------------------------------------------}
PROCEDURE PCIF_CLOSE ( VAR stat : ShortInt);
VAR
   dummy : Integer;
BEGIN
   { Finished, stop interfaces and clean up resources. }
   dummy := DcCleanUp("PCIFconfig");
   stat := 0;
END;
{-----------------------------------------------}
PROCEDURE PC_PCSTAT  ( VAR stat   : Shortint;
                           tag    : Shortint;
                           contwd : Shortint;
                           pc     : Shortint;
                       VAR buffr: ARRAY[1..512] OF Shortint);
VAR
   dummy : Integer;
BEGIN
   { Set up function parameters for machine status }
   Build_FSM_Status_Buffer;
   { Call DcSendToQ }
   dummy := DcSendToQ("host_init_input_q", msg_buff, buff_len, tag_no);
   { Call DcRdFromQ }
   dummy := DcRdFromQ("host_init_output_q", msg_buff, buff_len, wait,
                       tag_no);
   { Return values to application }
   Unpack_FSM_Status_Buffer;
END;
{-----------------------------------------------}
```

Host Initiated Message FSM Module

```
/* Code fragment written in HP DIS PSL language. */

/* structures used for reply to application */
       struct  AppReply    =(DeviceNo status key data);
```

```
/* structures used to match messages */
        struct  ReplyAck     = ( DLE ACK );
        struct  ReplyNak     = ( DLE NAK );
        struct  DataPacket   = ( DLE STX dst src cmd sts key data DLE
                                 ETX bcc );
        struct  ReqPacket    = ( device# function# address data);
Events
        /* Event:                Event Type, Event definition     */
        send_message:            request,  ReqPacket;
        process_reply:           response, DataPacket;
        unexpected_reply:        response, DataPacket;
        ACKed:                   response, ReplyAck;
        NAKed:                   response, ReplyNak;
        timeout:                 internal, TimePeriod >= TimeOutVal;
        process_noreply:         internal, RetryCount >= MaxRetry;
        retry:                   internal, RetryCount < MaxRetry;

State_table

IDLE:   send_message:
        /* Build the header */
        /* Send the message to the device */
        :WAIT;
IDLE:   unexpected_reply:
        /* Unexpected message, there must have been a timeout */
        /* Read the message from the queue and ignore */
        :IDLE;
WAIT:   process_reply:
        /* Process reply message */
        :IDLE;
WAIT:   timeout:
        /* Reset timeout */
        /* Re-send message */
        :RETRY;
WAIT:   NAKed:
        /* Read the message from the queue */
        /* Re-send message */
        :RETRY;
RETRY   :retry:
        /* Increment retry count */
        :WAIT;
RETRY:  process_noreply:
        /* Send no_message status to application */
        :IDLE;
```

Device Initiated Message FSM Model

```
/* Code fragment written in HP DIS PSL language. */

/* structures used for reply to application */
        struct  AppReply    = (DeviceNo status key data);

/* structures used to match messages */
        struct  ReplyAck    = ( DLE ACK );
        struct  ReplyNak    = ( DLE NAK );
        struct  DataPacket  = ( DLE STX dst src cmd sts key data DLE
                                ETX bcc );

Events
        /* Event:               Event Type, Event definition    */
        process_message:        response, DataPacket;
        ACKed:                  response, ReplyAck;
        NAKed:                  response, ReplyNak;
        host_init:              internal, cmd < 0;
        device_init:            internal, cmd > 0;

State_table

IDLE:   process_message:
        /* Reed the message from the queue */
        /* Send ACK to device */
        :IDENTIFY;
IDLE:   ACKed:
        /* Good, do nothing */
        /* Read the message from the queue */
        :IDLE;
IDLE:   NAKed:
        /* Bad, pass it on to host-init module */
        /* Read the message from the queue */
        /* Write a NAK to the host-init module */
        :IDLE;
IDENTIFY:       host_init:
        /* Send message to the host-init module for processing */
        :IDLE;
IDENTIFY:       device_init:
        /* Process device initiated message */
        /* Send message to application */
        :IDLE;
```

Configuration File

```
/* HP DIS  configuration  file */
PI_DEFINITIONS
    Runtime_Name = host_init;
    Port_Name    = port1;

    Runtime_Name = dev_init*;
    Port_Name    = port1;

PORT_DEFINITIONS
     Port_Name    = port1;
     Port_Major_# = 11;
     Port_Minor_# = 1;
     Baud_Rate    = 9600;
     Output_Queue = host_init_write_q;
     Input_queue  = device_init_read_q (dev_init);
     Unmatched_Queue = unmatched;

QUEUE_DEFINITIONS
     Queue_Name = host_init_input_q,  host_init_write_q,
                  host_init_read_q,   host_init_output_q
                  device_init_read_q, device_init_output_q
                  unmatched;

     Queue_Link
     host_init_input_q  >  host_init;
     host_init_output_q  <  host_init;
     host_init  >  host_init_write_q;
     host_init  <  host_init_read_q;
     device_init_output_q;  <  dev_init;
     dev_init  <  device_init_read_q;
```

Summary

This paper has shown a strategy for migration of factory control and monitoring programs that communicate with factory floor devices. This strategy uses a programmatic interface to transition to a finite state machine. The division of complexity was discussed. A simple model was developed using HP DIS to adapt PCTST and PCIF/1000 to an HP-UX computer. Simple code examples were shown to illustrate this model.

References

1. Denning, Peter and Jack Dennis, and Joseph Qualitz, "Machines, Languages, and Computation" Prentice-Hall, Inc., 1978.

2. Smith, Donald W. , "Finite State Machines for XModem" , Dr. Dobb's Journal, pp. 45, October 1989.

3. Garliepp, Kent, "Hewlett-Packard Device Interface System", Proceedings of the 1989 INTEREX HP Users Conference RTE, HP-UX, Workstations, September 11-14, 1989, pp 1015-1

4. Jue, Clemen, "A System to Develop Factory Floor Device Interfaces", Proceedings of the Third International IMS Conference, vol. 3, part 1, pp. 134-146, September 1987.

5. Litman, Felix, "Software Tools for Plant-Floor Connectivity", Automation, vol. 35, no. 4 , pp. 28, April 1988.

Life with MOMA: Using Starbase with the X Window System

John Pierce and Ruth Wright

Hewlett-Packard

2000 South Park Place, Atlanta GA 30339

Those who use the X Window System on HP-UX now have an extremely powerful way to present graphics—the merged Starbase/X Window System. The Starbase/X Window merge allows HP-UX users to have Multiple, Obscurable, Moveable, Accelerated (MOMA) windows, complete with 3D rendering and shading.

Such window-based graphics power comes from low-level software changes. Programmers do not need to know the intricate details of the changes—in fact, many Starbase programs require no changes to run in the merge environment. Yet in order to use the Merge, there are several new considerations. Because these considerations can have profound effects on your programs, it helps to know how the Starbase/X11 merged software works.

Architecture

Before HP merged Starbase and X11, a programmer could use the sox11 Starbase driver to display graphics in an X11 window. However, since "sox11" means "Starbase ON X11," Starbase calls were translated into corresponding

Xlib calls. This limited sox11 programs to X11's graphics capability. Figure 1 illustrates the architecture of Starbase On X11.

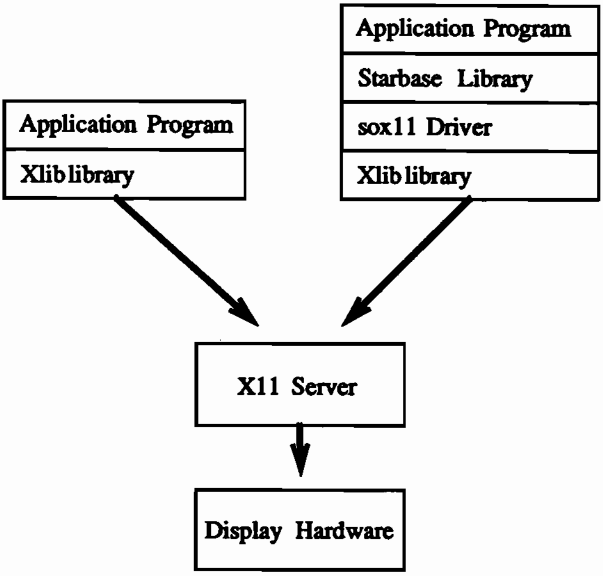


Figure 1. Starbase On X11 Architecture

To allow MOMA capability, the Starbase/X11 merge system allows Starbase programs Direct Hardware Access (DHA). With DHA, your program can use accelerated graphics hardware (the "Accelerated" in MOMA). DHA is implemented via a Graphics Resource Manager (GRM) that handles all graphics resource requests in an orderly manner. The GRM controls graphics resources such as shared memory, offscreen memory, color maps, and font glyphs (which are actually stored in offscreen memory). Programs do not need to take extra effort to be compatible with other X11 clients, because the GRM enforces cooperation between processes automatically.

The Starbase/X11 merge architecture is implemented at a low level, which means that upper level software programs (i.e. your application program) need no modification to work with GRM. In fact, both Starbase and X11 use GRM for all access to the display hardware. Figure 2 shows the organization of the Starbase/X11 merge. Since the GRM handles both X11 and Starbase resource allocation, there is no conflict between the two avenues of hardware access. Therefore, we may have multiple graphics windows (the "Multiple" in MOMA).

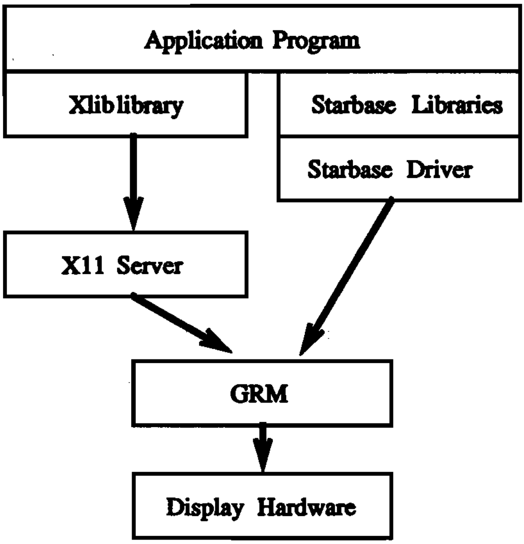


Figure 2. Starbase/X11 Merge Architecture

Although X11 and Starbase do not now fight each other for display resources, there is some ambiguity if each program has its own color map. HP has designed the Motif Window Manager (mwm) to resolve color map ambiguities between programs. Mwm has the responsibility to keep track of window focus events to update color maps.

Color Maps

Just what *is* a color map? A color map is a definition of the colors that the display hardware will have available, much like a painter's palette. Unlike a real palette, however, the color map has separate entries. You can't mix color map entries. The number of entries available in a color map varies from one display to another.

Both Starbase and X11 allow redefining color map entries. What happens if one program redefines the color map without notifying other programs? Without the Starbase/X11 merge, no process could safely change the color map. With the Starbase/X11 merge, each process may have its own color map, even though the display hardware can use only one at a time. How does the display hardware know which color map to use at a given moment? That's where mwm becomes important.

Mwm is a special X11 client, designed to manage windows. As such, it controls keyboard *focus*. The focus determines which window receives keyboard input. Since there is only one keyboard, mwm determines which window has keyboard focus either explicitly (by a click) or by pointer motion (by the pointer entering the window). The pointer focus policy is set as an mwm resource.

As long as mwm controls the keyboard focus, it also controls the color map focus. Since each process has its own color map, the GRM keeps software

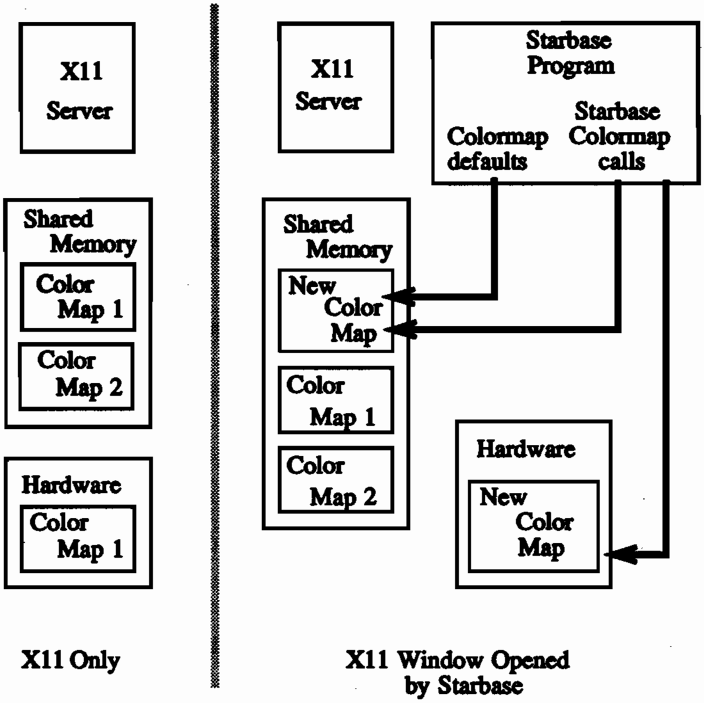


Figure 3. Color Map Control

color maps in memory. Mwm then directs the GRM to load a color map from memory to the display hardware whenever it changes focus to a window. Figure 3 shows how mwm allocates color maps and downloads them from software to hardware.

Although it's not shown here, all color map calls are submitted to the GRM, which in turn handles the details of actually allocating and managing the color maps. When mwm needs to change color map focus, it sends a request to the GRM to load a color map from shared memory into hardware.

Because the color map is valid for the window that currently has keyboard focus, only that window is guaranteed to have accurate color. Other windows that use the same color map index will use the current value of that hardware color map index. To guarantee that all colors remain constant, use xinitcolormap(1) to preallocate the X colors.

Mwm, the X11 server, and the GRM must use the available display hardware. To guide the Starbase/X11 merge system in how to use the graphics hardware, the X11 server works in one of four *operating modes*.

X11 Server Operating Modes

On some displays, the Image display planes have a device file that is separate from one for the Overlay display planes. How can the X11 server know how to use the separate Image and Overlay devices? By configuring the server in one of four operating modes.

Operating modes are configured in the file /usr/lib/X11/X0screens. (For full details on operating modes, refer to Starbase Graphics Techniques, HP P/N 98592-90078.) Entries in the X0screens file are tied to the display hardware by the name of the device file—e.g. /dev/crt for the Image plane device and /dev/ocrt for the Overlay plane device. To specify the operating mode, use the device file names in certain combinations. Here are the operating modes:

- Overlay Mode: The X11 server uses only the overlay planes.
- Image Mode: The server uses only the Image planes. If the hardware display does not have overlay planes, it must be used in Image mode.
- Stacked Screen Mode: The Image and Overlay planes act as two separate screens. Only one appears at a time. You may

shift from one to another by moving the mouse horizontally. When the pointer reaches the edge of the screen, it will "flip" to the other screen. This mode allows you to use another screen for displaying windows without another CRT.

- **Combined Mode:** The server uses both Overlay and Image planes at the same time.

Combined mode is probably the hardest to understand, since the server itself decides where to put a window. This decision is based on the requested *window depth* (the required number of planes). Window depth is related to the number of colors your window will be allowed to use. The formula for determining the number of colors is 2^n, where "n" is depth. For instance, if you decide you will use a depth of 1, you would only have two colors available, black and white; if you use a depth of 3, you would have 2^3, or eight colors available.

The server will use the minimum number of planes it needs to display a window. Since the Overlay device has a smaller plane depth than the Image device, the server puts "shallow" windows (those requiring less depth) in the Overlay planes. If the window needs more planes than the Overlay device can provide, the server puts it in the Image planes. If the window needs more planes than the Image device can provide, the server reports back an error to the program trying to create the window.

Graphics Window Creation

You may create a window in one of several ways. HP provides a special X11 client, xwcreate(1), to create a window. To specify a certain window depth, use the "*-depth n*" option. The "-r" option tells X11 to use *backing store*.

Backing store is the "Obscurable" in MOMA. With backing store enabled, X11 automatically restores graphics when windows are unobscured.

Xwcreate will make a special file that you later use in your Starbase gopen(3G) call. By default, the file is in the directory /dev/screen. Appendix B shows an example that uses xwcreate.

You may also create a window with the Xlib functions XCreateWindow(3X11) or XCreateSimpleWindow(3X11) to create a widget window. If you do, you will have the display and window IDs and may use any X11 call to manipulate the window. You may create a new color map or allocate colors within the current color map.

If the program uses OSF/Motif widgets, you may use the Motif function XmCreateDrawingArea(3X) to create a widget window, then get display and window IDs with XtDisplay and XtWindow. No matter how you create the window, mwm can move it around (the "Moveable" in MOMA).

If you use xwcreate(1), your program may be *window-dumb*. If you use X11 calls, your program will be *window-smart*.

Window-Dumb vs. Window-Smart

What is a window-dumb program? Simply, it is a Starbase program whose source code has no idea it's dealing with an X11 window. You just tell it where to draw and it draws. You still need to link in the window library (See Appendix B), but the code itself does nothing special to deal with the X11 environment. A window-dumb program may be used outside the X11 system, given the proper device file and Starbase device driver.

A window-smart program is one that knows it's dealing with X11. It creates its own window and may use X11 calls to manipulate the window. How does it know which device file to use? It uses the make_x11_gopen_string(3G) function.

Make_x11_gopen_string(3G) takes the display and window IDs and translates them into a string that gopen can accept as its first parameter (the *path* parameter). You may use make_x11_gopen_string(3G) directly in the gopen(3G) call.

Appendix A shows an example of a window-smart program that uses make_x11_gopen_string(3G) with an OSF/Motif drawing area widget. Because xwcreate(1) just builds a "free-standing" window into which Starbase can operate, it could not easily be used with the OSF/Motif widget library. By creating the drawing area widget (which is itself a window), Motif can handle the details of creation, mapping, and so forth.

Since make_x11_gopen_string(3G) requires both the display and window IDs, the program uses the X toolkit functions XtDisplay and XtWindow to retrieve the IDs.

Although the example is simple for illustrative reasons, the Starbase portion could be very complex. Direct Hardware Access (DHA) allows any Starbase feature available on the display hardware. If the program used the sox11 driver, it would be limited to X11 graphics features.

Note that if you use DHA from within the X Window System, you must compile the program with the "-lXwindow" ("search Xwindow library") option before the Starbase libraries sb1 and sb2. This searches the X11 DHA library to set up all access for cooperation through the GRM. The library search sequence is important. The Starbase Graphics Techniques manual shows the proper search sequence in Table 22-1.

Summary

Using Starbase with the X Window System gives you the option of either using "Starbase ON X11" or "Starbase WITH X11." Using Starbase ON X11 will allow you to run many existing (window-dumb) Starbase programs in the X11 environment by setting the proper device file and Starbase device driver, then recompiling them to link with the X window library. However, these programs will be limited to X11's graphics capability.

Using Starbase WITH X11 with Direct Hardware Access (DHA) allows your program to use all the graphics features of your display hardware—DHA is what gives MOMA capability. Low level software coordinates graphics resources so X11 and Starbase programs can safely coexist on the same display. Mwm selects the correct color map, based on current keyboard focus.

You may move existing Starbase applications into X11 by making your program window-dumb. If you write a window-smart application, you may also use X calls directly—giving your program the power of both Starbase and the X Window System.

Appendix A

Here is the Oneliner.c example from the Starbase Graphics Techniques manual, installed in an OSF/Motif window-smart program. The example was compiled with the command:

```
cc Oneliner.c -o Oneliner -ldd98731 -lXwindow \
              -lsb1 -lsb2 -lXhp11 -lXm -lXt -lX11 \
              -lPW -lm
```

The code is:

```
#include <stdio.h>
#include <X11/Xlib.h>
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/DrawingA.h>
#include <starbase.c.h>

/* Callbacks */
void     starbase_display();

int       fd;
Widget    wp_star_display;  /* Global for use
                               in callbacks */

main(argc, argv)
int   argc;
char **argv;
{

    Widget         toplevel;
    Arg            arg[2];
    int            n;


    /* Set up toolkit and connection to the server */

    toplevel = XtInitialize(argv[0],
             "Wp_starbase",   NULL, 0, &argc, argv);

    if (argc != 1) {
       fprintf(stderr, "unable to initialize\n");
       exit(1);
    }

    /* Workspace for displaying Starbase graphics */

n = 0;

XtSetArg(arg[n], XmNwidth, 400); n++;
XtSetArg(arg[n], XmNheight, 300); n++;

wp_star_display =
         XmCreateDrawingArea(toplevel,"star",arg,n);

XtManageChild(wp_star_display);

XtAddCallback(wp_star_display,
         XmNexposeCallback, starbase_display, NULL);

/* Realize the widgets */

XtRealizeWidget(toplevel);

XtMainLoop();
}



void starbase_display (widget, client_data, call_data)
Widget

   widget;
caddr_t  client_data;
caddr_t  call_data;

{
fd = gopen(
     make_X11_gopen_string(XtDisplay(wp_star_display),
                           XtWindow(wp_star_display)),
     OUTDEV,"hp98731",INIT);
```

```
/* Be sure that all Starbase operations are    */
/* complete, then sleep to allow user to see   */
/  line being drawn                            */

make_picture_current(fd);
sleep(5);
move2d(fd, 0.0, 0.0);
draw2d(fd, 1.0, 1.0);

/* Be sure that all Starbase operations are    */
/* complete, then sleep to allow user to see   */
/  the line after it has been drawn

   */

make_picture_current(fd);
sleep(10);

gclose(fd);

XCloseDisplay(XtDisplay(wp_star_display));
exit(0);
}
```

Appendix B

Here is the Oneliner.c example from the Starbase Graphics Techniques manual, in a window-dumb program. The example was compiled with the command:

```
cc Oneliner.c -o Oneliner -ldd98731 -lXwindow -lsb1
    -lsb2 -lXhp11 -lX11 -lm
```

The code is:

```
#include <starbase.c.h>
#include <stdio.h>


main(argc,argv)
int argc;
char *argv[];

{
int fildes;  /* file descriptor for gopen     */

if ((fildes = gopen(getenv("SB_OUTDEV"), OUTDEV,
                    getenv("SB_OUTDRIVER"),INIT)) == -1)
         exit(1);  /* exit if the gopen fails. */
move2d(fildes, 0.0, 0.0);
draw2d(fildes, 1.0, 1.0);

/* Sleep for 5 seconds so we can see the line     */
sleep(5);

gclose(fildes);

}
```

Once the code has compiled, create the window in which you want your starbase program to display with the xwcreate(1) command. For example:

```
xwcreate -geometry 500x500 -r -title starwin
```

will create a 500x500 pixel window named "starwin." How does gopen know how to access the "starwin" window? Xwcreate made a special file named "starwin" in the directory given in the WMDIR environment variable. By default, WMDIR uses /dev/screen. In this case, the special file's name is

/dev/screen/starwin

Now put the name of the special file in the SB_OUTDEV environment variable (remember that this is where the program will look for the Starbase device file name):

```
setenv SB_OUTDEV /dev/screen/starwin
```

for the C shell, or:

```
export SB_OUTDEV=/dev/screen/starwin
```

if you use the Korn shell.

The program also looks at the environment variable SB_OUTDRIVER to determine which Starbase driver to use. To use a Turbo SRX display with accelerator:

```
setenv SB_OUTDRIVER hp98731
```

in the C shell, or:

```
export SB_OUTDRIVER=hp98731
```

in the Korn shell.

Now that the window has been built and the environment initialized, run the program:

```
Oneliner
```

After you have finished running and viewing your program, destroy the window that you created with xwcreate:

```
     xwdestroy starwin
```

Notice that you have more work to do before running the program than with a window-smart program. Since the program knows nothing about windows, you must create the window yourself and set up some way for the program to access the window as though it were an actual graphics device.

International Application Development In PA-RISC

Jeffrey B. Caldwell and John W. S. Kwan

Hewlett-Packard Company
California Language Lab
19447 Pruneridge Avenue
Cupertino, California

1 Introduction

In today's market place, most software must be developed with the global market in mind if it is to be competitive. What this means is that an application may be used by users in a country other than the one in which it is written. Hence, software intended for the global market must be localizable. In other words, application programs must be written in a way that can be easily adapted for use in different countries or regions. Without any support from the environment under which the application is written, localization can only be achieved by modifying the application for each specific country. The result of such an operation is that multiple copies of the source must be kept and maintained, an expensive and error prone process. The solution to this problem is to design the application with globalization in mind. For example, messages and prompts should be stored in external files or catalogs. Language dependent functions (character comparison, up/downshifting, etc.) should be accomplished by external routines. This way, external files and catalogs can be translated, and the program localized, without rewriting or recompilation of the application. All this, however, cannot be done easily without support from the operating system and/or subsystems. Hewlett-Packard recognizes the importance of developing international software and provides an extensive set of tools and utilities to facilitate localization. These tools and functions are collectively known as Native Language Support (NLS).

In a programming context, Native Language Support refers to the facilities a programming environment provides to the software designer/writer so that application programs can be developed in such a way as to allow users to interact with the program in their native language, and which conform to the local customs, such as date/time formats and monetary symbols. In this paper, we will present many of the features provided by the PA-RISC operating systems, HP-UX and MPE XL, and programming languages, that help application programs to handle and process foreign (non-ASCII) characters, single byte as well as multi-byte.

2 PA-RISC Operating system Support

The MPE XL and HP-UX operating systems found on the Hewlett-Packard PA-RISC machines provide special support for the development and execution of programs intended to be shipped and run throughout the world. This section discusses support for NLS found on each of these operating systems.

2.1 MPE XL SUPPORT

MPE XL provides various support utilities, facilities and intrinsics to aid the application developer in creating easily localized programs. System Managers may use LANGINST (language install) to select and configure native languages to be supported on a system. NLUTIL is used to obtain the details of installed languages on a system. LANGINST and NLUTIL are fully described in the "Localizing/Customizing System Information Manual." Additional support is provided in the form of message catalog facilities, expanded 8-bit and multi-byte character sets to output non-ASCII characters, and a full set of NLS intrinsics to help the application developer create localizable programs. Message catalogs, character sets and NLS intrinsics are discussed below.

2.1.1 MPE XL Message Catalogs

2.1.1.1 What are Message Catalogs?

Message catalogs are special files that contain informative user messages to be output from applications. These messages are numbered and grouped into numbered sets. The messages can then be accessed by an application program and output to users. It is a central location that a program can keep all information that may need to be translated into another language.

2.1.1.2 Use of Message Catalogs

Using a message catalog to output messages is a convenient and efficient method to create a user interface. Message catalogs can be useful for more than localizing applications. An application developer may want to use a message catalog to:

- Ease efforts required to localize an application.
- Have prompts, error messages, informative messages, etc., that are intended to be output to users.
- Do not want to recompile the program each time messages are altered.
- Want easy access to the messages.

Messages catalogs allow the application developer to have a consistent and logical method of outputting messages to the user. The messages are separate from the code; therefore, localization (translation into other languages) is simpler. Only the message catalog needs to be translated and all the messages are in a defined area. Because the messages are apart form the code, it will not be necessary to retain the source code and recompile when a change is made to the catalog.

The use of message catalogs allows the application developer to create application message catalogs in a user's native language and access these messages programatically. Messages such as prompts, commands, error messages, or conventions for date and time can be stored in separate text files. As a result, the application developer can create and maintain files without changing the program itself.

2.1.1.3 Creating Message Catalogs

The MPE XL Application Message Facility contains the GENCAT.PUB.SYS utility program and the catalog intrinsics CATOPEN, CATCLOSE, and CATREAD.

GENCAT.PUB.SYS is the program that formats, incorporates your modifications, and unformats (expands) the message catalog files. GENCAT is menu-driven with a help facility which uses or creates four types of files:

- Source -- The application developer creates these files which contain sets of messages for the application.
- Formatted -- These files are created by GENCAT from the application developers source files. They contain a directory to the messages for quick access. The files are formatted in binary and are not intended to be edited. Formatted files are accessed by the application; the application then outputs the messages.
- Maintenance -- The application developer creates these files. Maintenance files contain corrections to source files, such as adding, deleting or replacing messages and sets.
- Collision -- When maintenance files are used to modify source files, GENCAT creates a

collision file. This file contains all the changes made to the source file by a given maintenance file. Using this collision file as a maintenance file on a previously modified source file will result in generating the original source.

Use of the GENCAT utility is fully described in the "Message Catalogs Programmer's Guide."

2.1.1.4 How To Access Message Catalogs

After a formatted message catalog has been created with the GENCAT utility, it may be accessed from an application with the system intrinsics CATOPEN, CATREAD and CATCLOSE. To read and output the messages, the catalog is first opened with the CATOPEN intrinsic and then accessed with the CATREAD intrinsic. Repeated accesses may be made once the catalog has been opened. An example of a command in Pascal that reads an opened catalog is

msglen := CATREAD (*catindex, setnum, msgnum, status, buff, buffsize, p1, p2, p3, p4, p5, msgdest*);

Only the first 4 parameters are required, *buff* through *msgdest* are optional. Upon completion of this statement *msglen* will hold the number of bytes contained in the message returned in *buff*. *Buffsize* is used to tell CATREAD the length of the buffer that has been supplied. *Catindex* refers to the catalog identifier received from CATOPEN (multiple catalogs may be open at any one time). *Setnum* and *msgnum* are used to tell CATREAD which message to access. *Status* will report if the CATREAD call resulted in an error, and, if so, what the error was. Supplying *msgdest* will cause the message to be automatically sent to the file number specified (e.g., 0 indicates $STDLIST). The remaining parameters, *p1* to *p5*, contain character strings to be inserted into the message at run-time. This provides the user with the flexibility to insert added information to a message such as a line number on which an error was found or a file name that could not be found.

Once access to a message catalog is no longer needed, the intrinsic CATCLOSE is used to close the catalog file.

2.1.2 MPE XL NLS Intrinsics

Application programs and Hewlett-Packard subsystems call NLS to obtain language-dependent functions and information for any language installed on a system. NLS uses the following categories of intrinsics (this information can also be found in the "Native Language Programmer's Guide" and "MPE XL Intrinsics Reference Manual").

Category	Intrinsic	Description
Information Retrieving	ALMANAC NLGETLANG NLINFO	Returns numeric data info. Returns current language. Returns language dependent info.
Character Handling	NLCOLLATE NLKEYCOMPARE NLREPCHAR NLSCANMOVE NLSWITCHBUF NLTRANSLATE	Compares strings. Compares strings. Creates nondisplayable chars. Moves and scans char strings. Modifies character strings. Translates to/from EBCDIC.
Time and Date Formatting	NLCONVCLOCK NLCONVCUSTDATE NLFMTCALENDAR NLFMTCLOCK NLFMTCUSTDATE NLFMTDATE NLFMTLONGCAL	Converts time format. Converts date format. Formats date. Formats time. Custom formats date. Formats date and time. Formats long version of date.
Number Formatting	NLCONVNUM NLFMTNUM NLNUMSPEC	Converts native numbers to binary. Converts binary number to native. Returns number conversion info.
Application Message Catalog	CATCLOSE CATOPEN CATREAD NLAPPEND	Closes a message catalog. Opens a message catalog. Returns messages from a catalog. Appends a lang # to a file name.

2.1.3 MPE XL Available Character Sets

Hewlett-Packard NLS includes a variety of character sets to aid the application developer. Each of the character sets has support for the following which can be accessed by use of the previously mentioned intrinsics:

- Upshift and downshift tables.
- Collating sequence tables.
- ASCII-to-EBDIC and EBDIC-to-ASCII tables.
- Long date format.
- Short date format.
- Time format.
- Currency symbols.
- Currency descriptor.
- Positioning and spacing of the currency symbol.
- Positioning and spacing of the thousands separator.
- Decimal and thousands separators for numbers.
- Equivalents of YES and NO.
- The full weekday names.
- The abbreviated weekday names.
- The full month names.
- Abbreviated month names.
- The National Date table (where applicable).

A list of some currently supported languages is included in the table below along with the ID number of each language. This ID # is used to set the JCWs NLDATALANG and NLUSERLANG to inform the system which language is to be the system default language.

ID #	Language	ID #	Language
0	n-computer	1	American
2	Canada-french	3	Danish
4	Dutch	5	English
6	Finnish	7	French
8	German	9	Italian
10	Norwegian	11	Portuguese
12	Spanish	13	Swedish
14	Icelandic	41	Katakana
51	Arabic	52	Arabic-w
61	Greek	71	Hebrew
81	Turkish	201	Chinese-s
211	Chinese-t	221	Japanese
222	Japanese.ujis	231	Korean

n-computer is the standard default language. It corresponds to the 7-bit ASCII character set. Hewlett-Packard NLS is continually evolving and being updated to encompass new languages and regions within countries. This list of languages can also be found on the HP-UX operating system.

2.2 HP-UX SUPPORT

The HP-UX operating system, besides providing function calls to handle input/output as well as the collating of Native characters (both 8-bit European and 16-bit Asian), also supports utilities that create and access message catalogs; the building of the language specific collating tables (to be used by other NLS functions); and environment variables that can be used to establish the current locale under which a program is running. NLS related functions are part of the C language library and will be covered in more detail later. Here we will discuss some of the utilities that are available to the users in helping them to develop software for the international market.

2.2.1 Message catalog handling

HP-UX provides the "gencat" utility that creates a formatted message catalog from a file containing error messages in a Native language.

Example :

```
gencat catfile datafile
```

Datafile could be in any of the supported Native languages and can contain several sets of error messages. Each error message must start with the error number. The gencat utility will create the message catalog "catfile" which can be accessed by either the catget() or the catgetmsg() functions. With this message catalog facility, error messages no longer have to be part of the program.

2.2.2 Building language specific collating tables

Every language has its own unique collating sequence governed by both customs and usage. In addition, time formats and monetary symbols are also different. HP-UX provides the "buildlang" utility that builds the locale.def file that is used by the setlocale() function in the LIBC library.

Example

```
buildlang [ -n ] infile
```

Buildlang will read a "buildlang script" from infile and create a locale.def file in the appropriate directory. If the -n option is specified, the locale.def file will be saved in the current directory. The locale.def file that is created contains informations about the collating table, time format, monetary symbol as well as other language specific informations. the "buildlang script" is quite long and is omitted here. Interested readers can consult the buildlang(1M) manual directly.

2.2.3 Environment variables

Users of HP-UX can establish the current locale of their programs by using environment variables. These environment variables are pre-defined names that identify certain categories of NLS operation. The environment variables are LC_ALL, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC and LC_TIME. These can be set to a supported locale which is usually a language or country name. They are used in conjunction with the setlocale() function of the C language and will be discussed in detail in section 3.1.

3 NLS Support in PA-RISC Compilers

Hewlett-Packard PA-RISC compilers have been designed and enhanced to provide the application developer with tools and support intended to simplify the task of developing applications with the international market in mind. This section discusses these tools and provides some simple examples of ways to access NLS functionality in ANSI C, C++, Fortran 77, COBOL II, Pascal and Business Basic.

3.1 NLS Support in ANSI C

For the C language, the support of native languages is a little different from other languages. The American National Standards Institute (ANSI) specifically incorporated NLS into the language definition. In other words, ANSI C requires that the LIBC library provides functions that can handle non-ASCII characters independent of the operating system. It further divides the support into several categories and the user can dynamically specify the particular language he wants for a particular category. For instance, he can specify that collating be done in German while the monetary symbol is Austrian. This unique ability to provide different support for different categories makes ANSI C an ideal language for developing software for the international market.

The main support for NLS in C is provided by the setlocale() function. The setlocale() function in C has the following syntax:

```
char *setlocale (int category, const char *locale);
```

The "locale" is usually the name of a supported language (e.g., German). The "category" specifies that only a certain aspect is to be set to that locale and the rest left unchanged. Defined in the locale.h header are the following categories:

LC_COLLATE -- string comparisons are done using the collating sequence of that locale.

LC_CTYPE -- character handling and multi-byte functions are based on the characters of that locale.

LC_MONETARY -- monetary outputs will be formatted according to the customs of that locale.

LC_NUMERIC -- numeric output will be formatted according to the customs of that locale.

LC_TIME -- times, as returned by the strftime function, are formatted according to the customs of that locale.

LC_ALL -- all of the above.

By definition, when a C program starts, the default locale is the C locale. It is up to individual installations to define what the C locale is. Generally it is setup to handle ASCII characters. The locale can be changed, of course, by calling the setlocale() function.

The setlocale function can be called in several ways; it will return different values depending on the locale specified.

Examples:

```
setlocale (LC_ALL, "german");
```

will set all categories to "german". If for any reason setlocale fails, it will return a null pointer and the current locale of the named category is not changed. If the locale is a null pointer, the current locale of the named category to the program will be returned. For example:

```
setlocale (LC_MONETARY, NULL);
```

will return to the program the locale that is set for monetary processing. This query operation will not change the locale environment of the program.

The setlocale function can also be used in conjunction with environment variables. Instead of specifying a locale explicitly in the function call, environment variables with the same names as the categories can be set and setlocale will check the setting and act accordingly. To use the environment variable feature, call setlocale with an empty string for the locale parameter as follows:

```
setlocale (category, "" ) /* "" is two double quotes (") */
```

and the locale will be set according to the following scheme:

If "category" is any defined value other than LC_ALL, setlocale will set the locale to the value of the corresponding environment variable if it exists. If the environment variable does not exist, or is set to an empty string, the locale will be set to the value of the LANG environment variable. In other words, setlocale (LC_MONETARY, "") will set the monetary symbol to FF (French franc) if either environment variable LC_MONETARY or LANG is set to contain the value "french". However, if either environment variable exists (or is set to an empty string), then the locale for LC_MONETARY will reset back to the default C locale.

If the category is LC_ALL, then all categories will be set according to the scheme above. In other words, setlocale will go through all categories to set the respective locale according to:

i. the value of the corresponding environment variable or
ii. the value of the LANG environment variable or
iii. the "C" locale

The above scheme works very well in HP-UX. In MPE, however, NLS is traditionally supported by setting Job Control Words (JCW); in particular, the JCWs NLDATALANG and NLUSERLANG. To be compatible with other products of the MPE family, C/XL will also check JCWs in addition to the environment variables (CI command variables). Hence, for MPE XL, the language in effect after a call to setlocale with a null string is, for each locale category, as follows:

- if an equivalent 'LC_' CI command variable is set, set the language to that specified by this variable, ELSE

- if the CI command variable LANG is set, set the language to that specified by LANG, ELSE

- if the JCW NLUSERLANG or NLDATALANG is set:

    - for LC_COLLATE and LC_CTYPE set language to language specified by NLDATALANG

    - for LC_TIME, LC_MONETARY and LC_NUMERIC set language as specified by NLUSERLANG, ELSE

    - set language to the default C locale.

Only the named category's locale will be changed. All other aspects of the NLS environment will not be affected. All data files needed by setlocale() are stored in LIB.SYS. As of MPE XL 2.1, there are currently 24 supported locale files each about 3K bytes in size. Users are free to remove any of the data files that they do not need from this group.

Other NLS functions

Once the locale of the program is established (by using the setlocale function detailed above), there are several functions the user can call to perform various NLS related tasks. They are:

1. localeconv

```
struct lconv *localeconv (void)
```

This function returns a pointer to a structure lconv (defined in <locale.h>) which contains various informations about how a number is represented in that locale (e.g., currency symbol of the locale, etc.):

```
struct lconv {
        char *decimal_point;
        char *thousands_sep;
        char *grouping;
        char *int_curr_symbol;
        char *currency_symbol;
        char *mon_decimal_point;
        char *mon_thousands_sep;
        char *mon_grouping;
        char *positive_sign;
        char *negative_sign;
        char int_frac_digits;
        char frac_digits;
        char p_cs_precedes;
        char p_sep_by_space;
        char n_cs_precedes;
        char n_sep_by_space;
        char p_sign_posn;
        char n_sign_posn;
};
```

Example:

```
lconv *p;
setlocale (LC_MONETARY, "german");
p = localeconv()
printf ("%s",p->currency_symbol);
```

This code segment will print out "DM" (for Deutsche mark) which is the monetary symbol for Germany.

2. strcoll

```
int strcoll(char *s1, char *s2);
```

This function compares two character strings using the collating sequence of the current locale. This function should be used instead of strcmp() whenever setlocale() is called to ensure that the correct collating table is used.

3. strxfmr

```
size_t strxfrm (char *s1, char *s2, size_t n);
```

This function transforms character string s2 to character string s1 such that after the transformation, string s1 can be compared using the strcmp function. The result will be the same as if strcoll was used to compare the original string.

Multi-byte character support

To provide support for Asian characters, ANSI C also defined a new data type wchar_t (wide character type) whose actual length is implementation defined but must be long enough to hold a multi-byte character. In addition, a set of special multi-byte functions are provided to handle those native languages that require more than one byte to represent (e.g., Japanese). They are affected by the LC_CTYPE category of the current locale. They are:

Multi-byte character functions:

mblen	returns the length (in bytes) of a multi-byte character
mbtowc	converts a multi-byte character to a wide character
wctomb	converts a wide character to a multi-byte character

Multi-byte string functions:

mbstowcs	converts a multi-byte string to a wide character string
wcstombs	converts a wide character string to a multi-byte string

Example:

The following sample segment of a C function will read a multi-byte character from a file and converts it to a wide character.

```
#include <stdlib.h>
#include <stdio.h>

wchar_t get_multibyte_char(void)
/*
 * Returns a multi-byte character.  If EOF is encountered then
 * it is returned.  If an illegal mult-byte character is
 * encountered, 0 is returned.
 */
{
     wchar_t return_value=0;
     char    char_buf[MB_LEN_MAX];
     int num_bytes;
     int     i;

next_char:
     for (i=0;i<MB_CUR_MAX;i++)
      if ((char_buf[i]=getc(in_file)) == EOF) {
               if (!i) return EOF;
           ungetc(char_buf[i--],in_file);
           break;
      }
     char_buf[i] = NULL;
     if ( -1 ==
          (num_bytes=mbtowc(&return_value,char_buf,MB_CUR_MAX))){
      error();
      goto next_char;
     }
     else
          /* return any extra bytes back to the file   */
          for (int j = i; j > num_bytes; j--)
                    ungetc(char_buf[j-1],in_file);

     return return_value;
}
```

NLS Support in C++

NLS support in C++ is similar to that of ANSI C. C++ supports international multi-byte characters in line comments, block comments, double quoted string literals and single quoted character constants. In addition, the use of environment variables and NLS library routine calls are the same as in ANSI C.

3.2 NLS Support in Fortran 77

3.2.1 HP Fortran 77 NLS Compiler Directive.

HP Fortran 77 provides the $NLS directive which supports processing of foreign language text and data at run-time and compile-time. The syntax of the directive is

$NLS [LITERALS | COMPARE] [ON | OFF]

When a HP Fortran 77 program is compiled on HP-UX, the f77 driver option -Y must be included for NLS functionality. This causes the compiler to link in the library /usr/lib/libportnls. Not including this option on HP-UX may result in the aborting of the program at run-time. This option in not necessary on MPE XL.

Use of the $NLS LITERALS directive at the beginning of a Fortran 77 program will enable the compiler to differentiate between closing single or double quotes of a string literal and the ASCII representation of a single or double quote found in a multibyte international character. It further differentiates between the ! character (beginning of an embedded comment) and the ASCII representation of a ! found in a multibyte international character.

Use of the $NLS COMPARE directive at the beginning of a Fortran 77 program will cause lexical comparisons with the Fortran routines LLE, LGE, LLT and LGT to automatically use the system configured Data Language (NLDATALANG jcw on MPE XL and NLDATALANG or LANG environment variable on HP-UX) for comparisons. This provides the Fortran 77 programmer with a simple means to perform comparing and sorting of international strings without the need of external system routines.

Including any $NLS directive (other than $NLS OFF) will also enable native language I/O support of a compiled Fortran 77 program when it is run. A particular issue found in Fortran 77 run-time I/O is the reading of quoted strings as input to character array variables. Enabling run-time NLS for I/O causes special code to be executed which will differentiate between quotation marks and the ASCII representation of a quotation mark in a multibyte international character.

Use of the $NLS ON directive enables the functionality of both the $NLS LITERALS and $NLS COMPARE directive. The $NLS OFF directive turns off all Native Language Support in the HP Fortran 77 compiler. This is the default

The following is an example HP Fortran 77 program that will read in 10 character string values, sort them according to the system configured Native language and output the sorted list.

```
$NLS ON
      program FtnNls
      character*20 list(10), temp
      print *, "Enter 10 strings, each on a different line"
      do 10 i = 1,10
        read *, list(i)
   10 continue
      do 30 i = 1,10
        do 20 j = 1,i
          if (lge(list(i),list(j))) GOTO 20
          temp = list(i)
          list(i) = list(j)
          list(j) = temp
   20   continue
   30 continue
      do 40 i = 1,10
        print *,list(i)
   40 continue
      stop
      end
```

3.2.2 Access to MPE XL NLS from HP Fortran 77

All NLS intrinsics discussed in section 3.1 can be accessed by using the HP Fortran 77 system intrinsic declaration mechanism. For example, to determine the current system configured value for the native user interface language (found in the NLUSERLANG jcw), the following program would suffice.

```
program GetUserLang
system intrinsic nlgetlang
integer*2 status(2), userlang
userlang = nlgetlang(1,status)
if (status(1) .NE. 0) STOP 'Error returned from nlgetlang'
print *,'System configured user language =', userlang
stop
end
```

3.2.3 Access to HP-UX NLS from HP Fortran 77

All NLS routines discussed in section 2.2 can be accessed by directly calling them from a HP Fortran 77 program. It is important to note that the HP-UX libc routines expect parameters passed in to follow the conventions of C datatypes. For example, a Fortran string must be converted to an array of characters terminated by the null character (ASCII 0) before being passed to one of the libc routines.

The standard HP-UX NLS functionality is designed for use with the C programming language. If it is desired to use this type of support when developing in HP Fortran 77, creating a small library in C of NLS routines for an application to use may provide a simple interface that is easier to use that constantly converting Fortran 77 datatypes to C datatypes before each call. Furthermore, by using C, the standard system NLS structures and header files can be used. This prevents the need to look at the /usr/include files and reproduce Fortran 77 equivalents for the various structures already supplied to C. Inter-language calling conventions can be found in the HP Fortran 77 reference manual in chapter 8, "Interfacing with Non-FORTRAN Subprograms."

As of the release of HP-UX 7.0, the only common NLS functionality supported on both MPE XL and HP-UX is the MPE NLS. An exception is the NLS provided by the C language (HP-UX 3.1 and beyond), mentioned in section 3.1. MPE XL NLS is the default on MPE XL and is documented in PORTNLS(5) on HP-UX. Including the option -Y option to the f77 driver will automatically link in the library /usr/lib/libportnls.a which contains a set of library routines that perform miscellaneous NLS-dependent operations found on MPE. Automatic inclusion of this library with the -Y option is only performed by the HP Fortran 77 driver. In order to access this library from other languages, it is necessary to use the option -lportnls. Localizable programs written in Fortran 77 which make use of these routines can be written and run under HP-UX, and ported into MPE by a mere recompilation, and vice versa. It may be necessary to maintain slightly different code for each operating system (e.g., the routines should be declared as system intrinsics on MPE XL and as external routines on HP-UX). This can easily be accomplished with the $IF, $ELSE and $ENDIF HP Fortran 77 directives.

3.3 NLS Support in HP COBOL II/XL

3.3.1 HP COBOL II/XL NLS Compiler option

As of the MPE XL 2.1 release, HP COBOL II/XL provides the $CONTROL NLS compiler option which enables support for international (multi-byte or non-ASCII) characters in certain character operations. This option makes string comparisons sensitive to international character sets and allows the source to contain international characters. The syntax for the option is

$CONTROL NLS = [LITERALS | COMPARE | ON | OFF]

This compiler option can only appear once in a HP COBOL II/XL program; on the first line or in the info string.

Using the $CONTROL NLS = LITERALS compiler option enables the HP COBOL II/XL compiler to support the scanning of international characters in non-numeric literals during the compilation of a HP COBOL II/XL program. International characters may require up to 16 bits to be represented (the standard ASCII representation only requires 7 bits). This directive ensures that the integrity of all 16 bits of each character found in non-numeric literals is maintained.

Using the $CONTROL NLS = COMPARE compiler option causes the HP COBOL II/XL compiler to perform relation condition comparison of non-numeric operands to be sensitive to the character set (and its associated collating sequence) that is the system configured Data Language (NLDATALANG jcw).

Using the $CONTROL NLS = ON compiler option enables support for both non-numeric literals and native language comparisons. Using the $CONTROL NLS = OFF compiler option in a HP COBOL II/XL program disables support for NLS; this is the default.

With $CONTROL NLS = ON, editing will insert the appropriate single-byte **DECIMAL-POINT**, **COMMA** and **CURRENCY-SIGN**. **ACCEPT . . . FREE** requires the appropriate **DECIMAL-POINT** for numeric data.

Using any of the $CONTROL NLS compiler options (other that $CONTROL NLS = OFF) may likely decrease compile-time and run-time performance. This is a result of the added complexity in scanning a source program and the problems encountered in performing comparisons between data that cannot be sorted simply by its ASCII bit representation. The use of auxiliary collation tables is required to determine where a character falls in relation to the languages lexical ordering when a non-ASCII character set is used. Additional information for NLS in HP COBOL II/XL can be found in the HP COBOL II/XL Reference Manual and in the HP COBOL II/XL Technical Addendum (HP Part No. 31500-90011).

3.3.2 Access to MPE XL NLS from HP COBOL II/XL

All NLS intrinsic routines discussed in section 2.1 can be accessed by use of the HP COBOL II/XL Call Intrinsic statement. HP COBOL II/XL adds the keyword INTRINSIC to the Call statement format to specify that an intrinsic, not a program, is being called. MPE XL NLS intrinsics are standard intrinsics and can be called exactly like any other intrinsic. Refer to the HP COBOL II/XL Reference Manual for more information on calling MPE XL intrinsics.

3.4 NLS Support in Pascal

3.4.1 Pascal NLS Compiler option

HP Pascal provides the NLS_SOURCE option which allows a developer of a Pascal program the ability to include multibyte international characters directly in Pascal literal strings or comments. The syntax of this option is

$NLS_SOURCE [ON | OFF]$

Without the use of this option it would not be possible for the compiler to differentiate between a closing quote of a literal string and the ASCII representation of a closing quote that happens to be one of the bytes in a multibyte international character. Likewise, the close of a Pascal comment can appear in a multibyte international character. This option will guarantee that such a multibyte character will not be incorrectly interpreted as a the closing of a comment. The HP Pascal Programmer's Guide has more information on this directive.

3.4.2 Access to MPE XL NLS from HP Pascal

All NLS intrinsics discussed in section 3.1 can be accessed by using the HP Pascal intrinsic declaration mechanism. For example, to determine the current system configured value for the native data processing language (found in the jcw NLDATALANG), the following program would suffice.

```
PROGRAM GetDataInfo(output);

TYPE Nlstatustype = ARRAY [1..2] OF SHORTINT;

VAR  data:Shortint;
     status:Nlstatustype;

FUNCTION nlgetlang: Shortint; INTRINSIC;

BEGIN
   data := nlgetlang(2,status);
   IF status[1] <> 0 THEN
      writeln('Error returned from nlgetlang. ERR = ',status[1])
   ELSE
      writeln('The current configured data lang. is ',data);
END.
```

A complete description of all MPE XL NLS intrinsics can be found in the MPE XL Intrinsics Reference Manual.

3.4.3 Access to HP-UX NLS from HP Pascal

All NLS support routines discussed in section 2.1 can be accessed by directly calling them from a HP Pascal program. The HP-UX libc routines expect parameters passed in to follow the conventions of C datatypes and strings. Therefore, Pascal strings must be converted to packed arrays of characters terminated with the null character (ASCII 0) before being passed to these routines. An example that reads in two strings and outputs the order determined by the collating tables of the Japanese language follows.

```
PROGRAM JapaneseSort(input, output);
CONST LC_COLLATE = 1; {the definitions for the LC types are     }
                      {found in the file /usr/include/locale.h  }
TYPE  PasStrType = String [20];
      CStrType = PACKED ARRAY [1..21] OF Char;
      {Note the extra element to allow for the null terminator  }

VAR   res : Integer;
      pstr1, pstr2 : PasStrType;
      cstr1, cstr2, japanese : CStrType;

PROCEDURE setlocale  { See setlocale(3C) for description }
        (category : Integer; locale : CStrType); EXTERNAL;
FUNCTION strcoll     { See string(3C) for description }
        (s1,s2 : CStrType): INTEGER; EXTERNAL;

BEGIN
   { Set the locale for comparisons }
   { Valid Language types can be found in /usr/lib/nls/config   }
   japanese := 'japanese';
   japanese[9] := chr(0) {ASCII 0};
   setlocale(LC_COLLATE, japanese);

   writeln('Enter 2 strings on separate lines');
   readln(pstr1);
   readln(pstr2);

   { Convert the Pascal strings to C-like strings. }
   strmove(strlen(pstr1), pstr1, 1, cstr1, 1);
   cstr1[strlen(pstr1)+1] := chr(0) {ASCII 0};
   strmove(strlen(pstr2), pstr2, 1, cstr2, 1);
   cstr2[strlen(pstr2)+1] := chr(0) {ASCII 0};

   { Compare the two strings. }
   res := strcoll(cstr1, cstr2);
   IF res < 0 THEN
      writeln('In Japanese, 1st string is lexically less')
   ELSE IF res > 0 THEN
      writeln('In Japanese, 1st string is lexically greater')
   ELSE  writeln('The strings are equal');
END.
```

If the Pascal programmer wishes to use the same NLS routines found on the MPE XL operating system, the library /usr/lib/libportnls.a can be used by including -lportnls in a Pascal driver string. See portnls(5) for more information. The comments found at the end of section 3.1.3, "Access to HP-UX NLS from HP Fortran 77," also pertain to HP-UX development in HP Pascal.

3.5 NLS Support in Business BASIC (MPE XL)

In Business BASIC, certain built in function provide enhanced support for NLS programming such as *LWC$*, *UPC$*, *LEX*, *DATE$* and *TIME$*. These are in addition to the standard MPE XL operating system support. By using the Business BASIC command *INFO*, the current language number and the name of the language can be determined. The underlying native language can be changed with the *RUN* and *SCRATCH ALL* commands after the NLDATALANG jcw has been changed by either of the following:

```
> SYSTEM "setjcw NLDATALANG = 12"
> :setjcw NLDATALANG = 12
```

3.5.1 Business BASIC support functions

LWC$

Purpose: LWC$ is a string function that returns a string with all characters shifted to lowercase.

Syntax: LWC$(string_expression [,native_language_number])

Where string_expression is Business BASIC string or string expression and native_language_number is a numeric expression.

Values and meanings for native_language_number:

native_language_number	meaning
no argument	Use NATIVE-3000 conversion
-1	Use the downshift table appropriate to the language denoted by the underlying native language number.
>= 0	Use the downshift table appropriate to the language denoted by native_language_number.
< -1	illegal

Example:

```
        10 ! Line 20 uses the Spanish conventions for downshifting.
        20 PRINT LWC$("Mi nombre es Enrique",12)
```

UPC$

Purpose: UPC$ is a string function that returns a string with all characters shifted to uppercase.

Syntax: UPC$ (string_expression [,native_language_number]) Values and meanings for string_expression and native_language_number are the same as in *LCW$* above.

Example:

```
        10 ! Line 20 uses the Spanish conventions for upshifting.
        20 PRINT UPC$("Mi nombre es Julio",12)
```

LEX

Purpose: This allows the comparison of two strings in a Native Language dependent manner. It is useful for sorting and/or alphabetizing.

Syntax: LEX(string1, string2 [,language-number]) Where string1 and string2 are Business BASIC strings or string expressions and language-number is a number or numeric expression which designates the native language whose collating sequence should be used (the collating sequence values can be found in section 2.2).

The value returned by the LEX function will reflect the order in which the two strings are lexically ordered. -1 is returned if string1 < string2, 0 is returned if string1 = string2, and 1 is returned if string1 > string2

Example: Since ch and c are both valid letters in the Spanish alphabet, sorting a string with ch and c can have different results if the Native Language has been set to Spanish instead of English or N-computer.

```
            100 print lex("coin","choice",0)  ! Prints 1
            120 print lex("coin","choice",12) ! Prints -1
```

DATE$* and *TIME$

Purpose: These are Business BASIC functions that will return string representations of the current system time and date. It is possible to cause these functions to return a string that is adjusted to represent the current Native Language configuration of the system by including an optional argument.

Syntax for Native Language Support of DATE$ and TIME$:

DATE$ (num_expr)

Values and meanings for num_expr:

value	Format returned
-2	*(yyyy/mm/dd)*
-1	System configured format (NLDATALANG)
0	U.S. format *(mm/dd/yy)*
1	European format *(dd.mm.yy)*
other	format specified by Native Language *num_expr*

TIME$ (num_expr)

Values and meanings for num_expr:

value	Format returned
-1	System configured format (NLDATALANG)
>= 0	format specified by Native Language *num_expr*

3.5.2 Access to MPE XL NLS from Business BASIC

All NLS intrinsics discussed in section 3.1 can be accessed from Business BASIC by using the standard Intrinsic and Global Intrinsic statements found in Business BASIC. A complete description of intrinsic support is documented in the "HP Business BASIC Reference Manual."

4 Conclusion

We have presented here many of the tools currently available to the users of the PA-RISC systems to develop localizable software. The concept of NLS support is a relatively new one among software developers; however, it is one that is gaining importance. There are several international organizations that are involved with various aspects of NLS works ranging from requirements specification to the standardization of character set representations. One such organization is the International Standards Organization (ISO) which has published standard character sets for several European languages. Both the American National Standards Institute (ANSI) and the IEEE's Standard, Portable Operating System Interface for Computer Environments (POSIX), have included NLS into their requirements. Hewlett-Packard is an active participant in all of these organizations and is committed to follow the standards as they evolve. NLS support in the PA-RISC systems will continue to be improved and expanded to provide users with the latest tools available to satisfy the international market.

APPLICATIONS NETWORKING IN THE ANALYTICAL ENVIRONMENT

Debbie Szecsei

Scientific Instruments Division
Hewlett-Packard Company
1601 California Avenue, Palo Alto, CA 94304

INTRODUCTION

The hottest issue today with any computer company, including HP, is networking. Everyone wants to know that their computers can talk to other people's computers.

But is that really enough? Customers need software solutions in addition to transferring data between computers. And that means taking a look at the applications that customers are using, and creating software that takes advantage of networking to save the customer time and money.

This paper will focus on Applications Networking between an HP-UX based HP 9000 and an RTE-based HP 1000 (Rev. 5.1 or greater). [1]

CUSTOMER OVERVIEW

Our customers are analytical chemists who use instruments such as Mass Spectrometers (MS), Gas Chromatographs (GC), and Liquid Chromatographs (LC) in their laboratory.

At Scientific Instruments Division we provide analytical instrument control, data acquisition and data analysis systems. These systems run on the RTE, Pascal WorkStation, HP-UX and MS-DOS environments. On the HP-UX, Pascal and MS-DOS operating environments, the analytical software systems are commonly referred to as ChemStations.

We have a large installed base of customers with Pascal WorkStations and HP 1000's who would like to buy HP-UX but who don't want to abandon their existing technology. They want to tap into the power of HP-UX and, at the same time, protect their investment.

There are several reasons for the growing interest in HP-UX: it is a true multi-tasking system which can simultaneously transmit and receive multiple networking requests. Networking software (ARPA services) is standard when you buy HP-UX. HP-UX also ships with X-Windows, an industry standard application that is designed to work within a networked environment.

[1] The solution is published in the Application Note "HP-UX ChemStation —> LABSAM; Report Translation and Automation", developed by the Unified Laboratory at SID. (P.N. 23-5953-6378). Available through HP's Literature Distribution Center.

With multiple windows on a single HP-UX terminal running X-Windows, you can communicate with more than one machine at the same time. For example, one window can be logged onto a Pascal WorkStation, another window logged into a Laboratory Information Management System (LIMS), and another window talking to a remote HP-UX ChemStation. (Figure 1.)

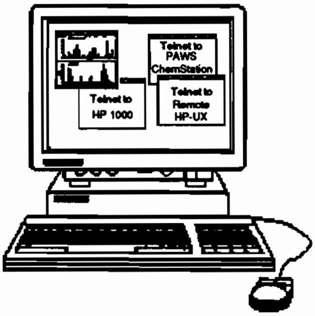


Figure 1. HP-UX Terminal With Multiple Windows

Figure 2 shows a possible combination of analytical data systems connected over a LAN. An HP-UX ChemStation, a Pascal WorkStation, a DOS ChemStation, and an RTE-A HP 1000 are shown with their associated protocols for communication.

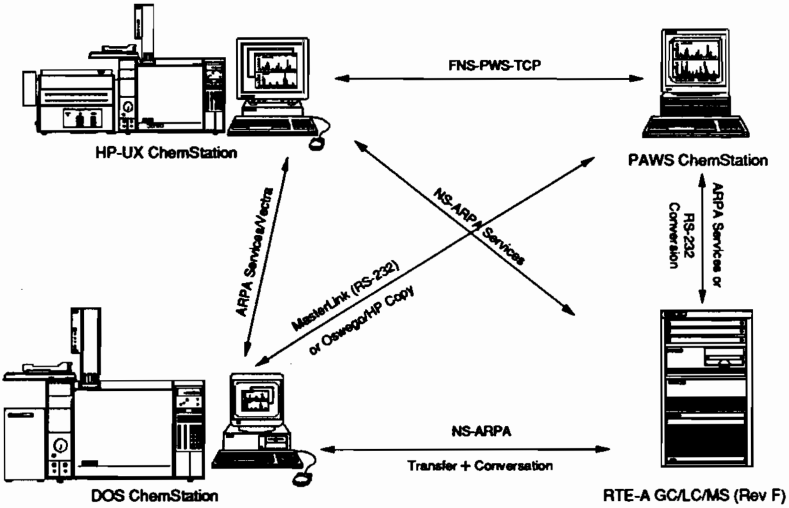


Figure 2. Analytical Data Systems in the Unified Laboratory

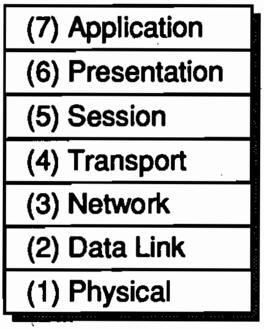
Note that most of the systems communicate via ARPA services, and in the case where the operating system does not have ARPA service capability, NS is used. This way, the "older" computers like the HP 1000 are able to communicate with the "newer" technology, HP-UX and MS-DOS. In addition, standard networking protocols (i.e. TCP/IP, IEEE 802.3) are used to connect the various systems on the LAN.[2]

[2] For additional connection possibilities, please see The Connectivity Matrix (P.N. 23-5956-4107), developed by the Unified Laboratory at SID. Available through HP's Literature Distribution Center.

NETWORKING MAGIC

Customers frequently hear about networking these days, but many still think it's magic. Or worse, they think networking is too complicated and expensive to implement. Many sales representatives have seen impressive networking demos, but many of them aren't sure how to bring a networking solution into a real-life laboratory.

Networking is not magic. Standard protocols are used across operating systems so that applications written by different vendors can communicate with each other. The technology is at a point where we can almost take it for granted. What's needed to make networking *meaningful* is software that connects the various applications across product lines. Consider the OSI Network Model in Figure 3.



(7) Application

(6) Presentation

(5) Session

(4) Transport

(3) Network

(2) Data Link

(1) Physical

Figure 3. International Standards Organization Model of Open System Interconnection (ISO/OSI)

Layers 1 and 2 (Physical and Data Link layers) refer to the actual LAN card and cables that plug into the computer.

Layers 3, 4 and 5 (Network, Transport and Session layers), are comprised of firmware; an interface that corresponds to the actual movement of data from one computer to another. These layers correspond to the TCP/IP protocol that most people are familiar with.

Layers 6 and 7, (Presentation and Application layers), refer to the software applications that make it possible to translate data into meaningful information. These two layers corresond to NS/ARPA services. The NS/ARPA programs most people are familiar with include DSCOPY, FTP and Telnet.

The hardware/software in the OSI model already exists. The challenge for Applications Networking is to provide intelligent application compatibilities between software products so that the tangible benefit of networking, i.e. increase in overall productivity, is realized. This would essentially be an eighth layer above the Application Layer in the OSI model.

The Unified Laboratory Group at SID develops networking products belonging to this conceptual eighth layer, and one of the solutions we've come up with is discussed in the Case Study below.

CASE STUDY

It is currently possible to connect the HP-UX ChemStation and the HP 1000 computer via a LAN (Local Area Network). There are several things customers want to do with data across the two systems:

- Transfer a report file from the HP-UX ChemStation to a LIMS on an HP 1000. Have the report automatically copied into the database so the customer doesn't have to enter the results manually.
- Automate the above process so that the report can be transferred as part of the customer's routine software tasks.

The standard ChemStation software is not able to automate the report file transfer from HP-UX to the HP 1000, and even if the file transfer could be automated, the software on the HP 1000 cannot automatically copy the report into the database.

These tasks require Applications Networking software to be written.

DEFINITIONS

When discussing the HP-UX ChemStation and HP 1000 systems, some terminology needs to be defined:

Data File: The raw data (binary format) that is acquired by the instrument on the HP-UX ChemStation.

Report File: An ascii file that is generated after analyzing the raw data on the HP-UX ChemStation.

Method: A series of tasks for the HP-UX ChemStation to do (Data Acquisition, Data Analysis, etc.)

Macro: A "pseudo-programming" language used to automate tasks within the HP-UX ChemStation environment. Chemists are able to write their own macros.

LABSAM: Hewlett-Packard's implementation of LIMS on the HP 1000. LABSAM is used to organize large amounts of sample information.

Daemon/Mapper: A program that copies information into LABSAM.

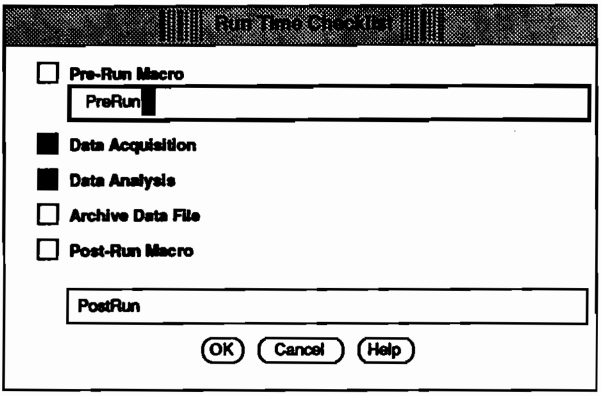
Definition File: A template file that tells the Daemon/Mapper program what information to copy into LABSAM.

THE TASK

The customer acquires data via an instrument, which is controlled from the HP-UX ChemStation. The customer sets up a method, which tells the ChemStation software what to do:

Acquire a data filę and analyze it

The run-time checklist for a method that does this is shown in Figure 4. Notice that Data Acquisition and Data Analysis are both selected.



Run Time Checklist

□ Pre-Run Macro

PreRun

■ Data Acquisition

■ Data Analysis

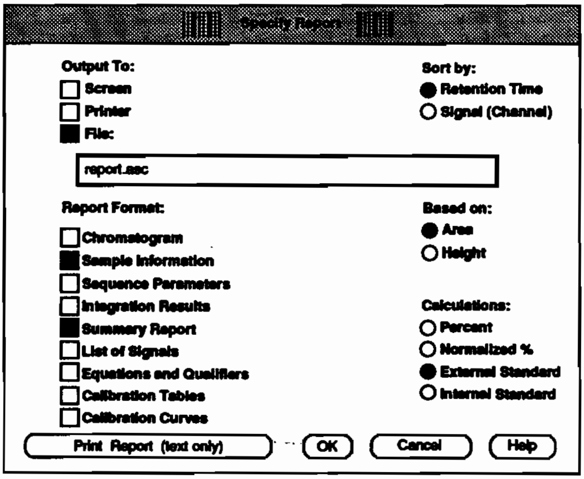
□ Archive Data File

□ Post-Run Macro

PostRun

Figure 4. Run-Time Checklist in a Method

The customer tells the HP-UX ChemStation to generate an ascii report on the Specify Report screen (Figure 5.) Notice that the checkbox File: is selected, and the name of the report is "report.asc".



Specify Report

Output To:
□ Screen
□ Printer
■ File:

report.asc

Sort by:
● Retention Time
○ Signal (Channel)

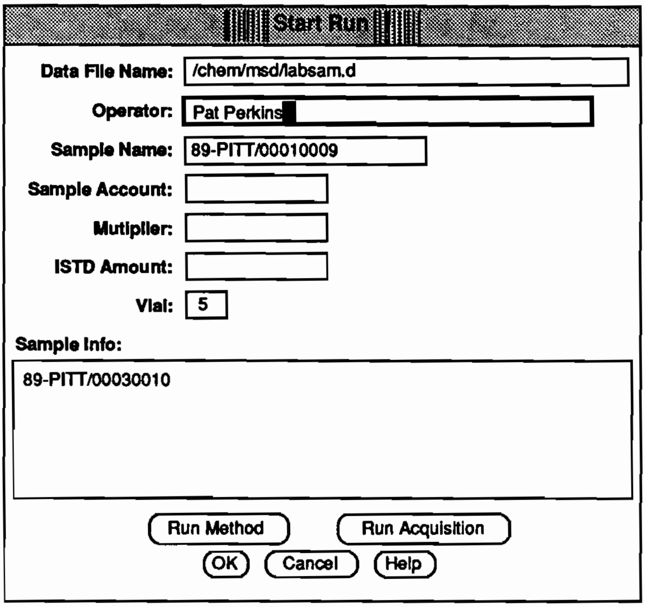
Report Format:
□ Chromatogram
■ Sample Information
□ Sequence Parameters
□ Integration Results
■ Summary Report
□ List of Signals
□ Equations and Qualifiers
□ Calibration Tables
□ Calibration Curves

Based on:
● Area
○ Height

Calculations:
○ Percent
○ Normalized %
● External Standard
○ Internal Standard

Figure 5. Specify Report screen

The customer runs the method by calling up the Start Run panel. They specify the name of the data file, and the "key", which will identify the sample in the LABSAM database, is entered in the Sample Info field (Figure 6).



Start Run

Data File Name: /chem/msd/labsam.d

Operator: Pat Perkins

Sample Name: 89-PITT/00010009

Sample Account:

Mutiplier:

ISTD Amount:

Vial: 5

Sample Info:

89-PITT/00030010

Figure 6. Start-Run Panel on the HP-UX ChemStation

When the user clicks the "Run Method" button on the Start Run panel, the data is acquired and then analyzed. After analysis is complete, a report is generated. (Figure 7).

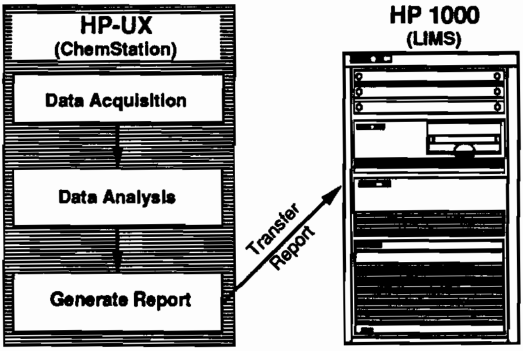


Figure 7. Data flow from HP-UX to the HP 1000.

At this point in the data flow, the report is automatically copied over to the HP 1000, using DSCOPY from a macro (Appendix A). Note that in the macro, the user has to customize 3 pieces of information:

machine name
user name
password

This customization only has to be done once, when the user first types in the macro. From then on, the user just clicks "Run Method" on the Start Run panel and the report will automatically be copied to LABSAM.

Not only can the user forget about remembering the machine name, user name and password each time a file gets copied, but the user also doesn't have to remember the syntax for the DSCOPY command!

The best part about this solution is that it can be done in an automated fashion. The HP-UX ChemStation can be acquiring a sequence of data files, and each one can be copied over the network, unattended.

On the target system, LABSAM, there is even less human-interaction. Once the initial configuration to the database and Daemon/Mapper has been done, the user only has to be sure that the Daemon/Mapper program is running in order to get the report into the database[3].

DATA FLOW

When a customer uses our Applications Networking software, several things happen. The user acquires and analyzes their data on HP-UX, then *automatically* transfers the resulting report over to the HP 1000. Once the report is on the HP 1000, Daemon/Mapper *automatically* copies the report into the database. After the report is copied into the LABSAM database, it can be accessed on several LABSAM screens (remotely from HP-UX using Telnet or else logged on to the local LABSAM system).

It is important to note that the technology was already in place to do what the customer wanted. The task itself appeared too difficult, and customers were unsure of how to go about doing it. With Applications Networking software, the task no longer seems difficult.

With the click of one mouse button, the customer is now able to:

- Interface between two unique software applications (i.e. acquire data on one system and store it in a database on another).
- Use HP-UX networking transparently
- Use HP-UX X-Windows to perform more than one task at a time
- Guarantee the connectivity of existing hardware with the latest HP-UX hardware and software

[3] Please see the Application Note for details on customizing the LABSAM database and the Daemon/Mapper program.

WHAT WE PROVIDED

- ◆ Applications Networking Software which included:
    - A macro (Appendix A) to transfer the report from the HP-UX ChemStation to LABSAM. The user customizes the macro for the target machine, user name and password. In addition, the macro can be customized for the type of data being tranferred.
    - A method which is set up to acquire the data, analyze it, generate the report and then copy the report to LABSAM.
    - A Defintion File (Appendix B) for the Daemon/Mapper program on LABSAM. This Definition File defines what information in the report should be stored in the database.

      In addition, this definition file can be modified by a customer who wants to extract different data from the report. (Customers who have LABSAM typically understand definition files.)
- ◆ An Application Note with step-by-step instructions for implementing the solution.
- ◆ A Trouble Shooting Guide to help the users if the file transfer fails for some reason.
- ◆ Customer Demos

We demonstrated the solution to customers who wereconsidering buying HP-UX and already had RTE systems. They wanted to see for themselves that HP is committed tonetworking the two systems together. This solution also works for MS-DOS ChemStations to LABSAM with only minor modifications.

DEVELOPING NETWORKING SOFTWARE

Applications Networking developers should think of building a conceptual eighth layer on top of the OSI Model. This layer will address the problems of providing application compatibilities between disparate technologies over a networked environment.

Based on our experiences at SID, an application specifically designed to bring programs together across different operating systems should follow these principles:

- Develop on top of networking standards; don't re-write basic networking software
- Create a graphical user-interface, whenever possible
- Shield the customer from cumbersome networking commands

- Automate the solution, whenever possible
- Provide an Application Note which contains adequate documentation to do the task, and which references other documentation for additional details. This can be invaluable in helping novice users get their network and software up and running.

SUMMARY

In order for our existing customer base to appreciate the true power of networking in their laboratories, we need to provide elegant Networking Application software to connect applications across product lines. It is not enough to just physically connect computers together, users need to be able to accomplish a meaningful task once the computers are connected.

The Case Study in this paper discusses the automatic transfer of a report from an HP-UX ChemStation into a LIMS database with ease. The customer doesn't have to be burdened with remembering the machine name, user name and password for each file transfer, or the syntax of the networking commands; all that information is encapsulated into a macro.

By automating the process, file transfers can be done within a sequence and by unsophisticated users. By using X-Windows, the process is executed with the click of a mouse button. Since HP-UX is multi-tasking, the computer can be doing other things while performing the network file transfer. The ChemStation can acquire another data file, or even be logged onto another computer system, all at the same time. The entire solution uses standard network protocols to connect HP-UX to the HP 1000.

This is only one example of an Applications Networking solution. Please let your HP representative know if you have other networking needs that are not being met. We are always interested in customer's ideas for increasing productivity in the lab, via networking. That's how the best products remain the best.

Appendix A. Labsamcopy Macro

```
 This macro will use DSCOPY to copy a report file from the HP-UX ChemStation
to the
! <<search directory>> on LABSAM. The <<LABSAM machine name>>,
! <<LABSAM user name>> and <<LABSAM password>> are all hard-coded.
!
name labsamcopy
  LOCAL machine$, user$, passwd$, srcfile$, dstfile$, runstring$
  on error E
  machine$ = "<<LABSAM machine name>>"
  user$ = "<<LABSAM user name>>"
  passwd$ = "<<LABSAM password>>"
  search$ = "<<search directory>>"
!
  cpstop ON
!
  genfilename
!
  srcfile$ = FILENAME$ +"/" + "report.asc"
  dstfile$ = search$ + "/" labsamfname$
  dst$= machine$ + "#" + user$ + "/" + passwd$ + "#" + dstfile$
!
  runstring$ = "dscopy -r " + srcfile$ + " " + dst$
  print "runstring is: " + runstring$
  print "Copying " + srcfile$ + " to LABSAM system: " + machine$
!
  execwait runstring$
!
  print "Copy finished"
E: reporterror
  cpstop OFF
  return
! ********************************************
! This macro uses the Data Analysis variable FILENAME$ to generate the name
of the
!  report on the HP1000. The path must be stripped off of the data file name, and
the
! ".d" must be removed. Then the .UCS suffix is appended, to get the final result.
! The <<data filename>> is put in the string variable labsamfname$.
!
! EXAMPLE:   if FILENAME$ is:  /chem/msd/demoscan.d
!            labsamfname$ will be:   demoscan.ucs
!
name genfilename
  LOCAL lowrange, highrange, slash$, tempname$
  tempname$ = FILENAME$
  slash$ = "/"
  on error E
!
  labsamfname$ = tempname$
  lowrange = instr(labsamfname$,"/")
!
  while lowrange <<>> 0
```

Appendix A. Labsamcopy Macro *(continued)*

```
    tempname$ = labsamfname$
     labsamfname$ = tempname$[lowrange+1:len(tempname$)]
     lowrange = instr(labsamfname$,"/")
   endwhile
   highrange = instr(labsamfname$,".")
   if (highrange <<>> 0) then
     labsamfname$ = labsamfname$[1:highrange-1]
   endif
!
! Append the .UCS suffix to the filename
!
   labsamfname$ = labsamfname$ + ".UCS"
E: reporterror
   return
```

Appendix B. Definition File for HP-UX ChemStation Report

```
** Define '*' as comment character

**  Is delimiter for in memory array
** List on lu 6 with 58 lines per page
6 58
** MO <<directory>>  or DE  or NO    (move, delete files without errors)
MO /<<search directory>>/NOERROR
** MO <<directory>>  or DE  or NO    (move, delete files with errors)
MO /<<search directory>>/ERROR
** N means continue if idc definiton is missing
Y
** There are 2 test names in the IDC match list:
2
************** IDC Match List **********************
Cl-diphenyl       \Cl'diphenyl
methyl palmitate   \Methyl'palmitate
*
* LABSAM doesn't accept IDC's with hyphens or spaces in them. I define the
* IDC with underscores in the database, and map it here in the match list.
*
***/<<search directory>>/def/hit.UCS

** End of fixed part
** Definitions in the next block may be in any order

** define an audit comment (optional). Define only one audit comment!
** A,"<<comment>>"
A "Result was overwritten from mapper def file DEF.UCS"

** IDC key definitions. The key always appears on line 2.

2 2 "$KEY1" 1 16
2 2 "$KEY2" 18 33

* Definition lines for Header info
* Operator                    C20 format
4 $ "OPERATOR" 17 36 C"Operator:" 6
* Date Acquired               C10 format
4 $ "DATEACQ" 21 33 C"Date Acquired:" 6
* Time Acquired               C15 (HH:MM:SS)
4 $ "TIMEACQ" 35 45 C"Date Acquired:" 6
* Sample Name                         C30 format
4 $ "SAMPLE'DESCRIPT" 20 49 C"Sample Name:" 6
* Misc Info                   C72 format
4 $ "COMMENT1" 18 89 C"Misc Info:" 6
* Bottle Number                       F30
4 $ "BOTTLE'NUM" 22 24 C"Bottle Number:" 6
* Repetition Number                   F30
* The entire criteria "Repetition Number:" is too long for C criteria ...

4 $ "REP'NUM" 48 50 C"Repetition Num" 28
** IDC startline,endline,startname,endname,startvalue,endvalue,(Options)
** UCSMETH is an LAS IDC
```

Appendix B. Definition File for HP-UX ChemStation Report *(continued)*

```
12 12 "UCSMETH" 53 56 1000
* Definition lines for Table Entries
* Retention Time, if compound name is blank and a peak was found
* 26 $ UCSMETH'T" 20 25 C"    " 40 VCN"****" 1
* Signal Description if compound name is blank
* 26 $ UCSMETH'H" 27 32 C"    " 40
* Area if compound name is blank
* 26 $ UCSMETH'A" 59 66 C"    " 40
* Amount if compound name is blank
* 26 $ UCSMETH'C" 67 73 C"    " 40
* Enter compound name and amount, if a peak was found
20 $ 38 56 67 73 VCN"    " 68
** define replacment for invalid results (illegal format)
** R "<<replacement string>>" ["result string to replace"]
R "#no res." " Found"

** validation check:  V linenumber validationstring startbyte
```

Using SAR to Examine System Performance

Wayne R. Asp
Hewlett-Packard Company
2025 West Larpenteur Ave.
St. Paul, Minnesota 55113

How well is my system performing? Why is the system so slow? How much CPU do I have left? These are all system performance questions that most HP-UX System Administrators are asked to address.

This paper will first examine what constitutes system performance. Following this examination, a system performance tool within HP-UX called SAR will be presented as a vehicle to address certain system performance issues. SAR capabilities include both collecting and reporting kernel level performance data from HP-UX. Next, strategies for interpreting SAR data are presented. Lastly, other currently available performance tools on the HP-UX operating system are contrasted with SAR.

Performance, with regard to a system, means to provide optimum performance when executing the tasks required of the system. While this statement may seem very vague, it is realistic in that true system performance problems are also vague and very often difficult to isolate and address. The global term "system performance" can be defined as anything from resource utilization to application efficiency to workload balancing. To really understand system performance one must first understand what is meant by system performance.

System performance can be divided into four main areas as follows:

Resource Utilization.
This involves determining specifically how software and hardware resources are allocated and used within the system. Generally, tools are used (such as SAR) to report operating system parameters and resources as the system is executing. By examining and comparing this resource utilization data from the system, a System Administrator (or a performance consultant) can determine where bottlenecks and other restrictions of resources exist. Since resource utilization data generally includes information on executing processes and I/O loads, some workload balancing can also be performed.

System Configuration.
How are the current hardware and software components configured? Any configuration contains implications with regard to performance. Some system configuration issues will become apparent when the resource utilization of the system is examined, but this is not always the case. Certainly, resource utilization tools cannot identify directly system configuration issues and problems, but can direct the performance investigation towards system configuration.

Capacity Determination and Planning.
What will happen if another application or user is added to the system? This is a typical question in the capacity planning area. The capacity of the system to perform all tasks requested of it will obviously diminish as the number of tasks grows larger and system resources are allocated to their maximum. To provide planning for future capacity and growth, one must examine the current workload on the system and determine how that work load can be balanced and additional capacity realized. In many instances this will mean collecting current performance data for the system and projecting that data into future task requirements. Benchmarks are a capacity planning tool.

Application Tuning.
Applications consume system resources. How many and exactly what resources does a specific application consume? To address this, one must actually look within the application and determine where the bottlenecks and resource issues reside within the application itself. While the other three performance areas address issues in a global system sense, application tuning refers strictly to how the application is designed and executes internally.

Categorizing system performance into these four areas allows one to differentiate performance issues and assess their impact on specific areas of the system. To generally state, for instance, "the system is slow" does not provide enough information for one to pursue the issue. However, to state "the system is slow when 16 users are logged in" will provide enough information for an investigation to begin in the area of resource utilization, and specifically, users logged into the system.

In fact, optimum system performance is a combination of all four performance areas described previously. Certainly, if a resource utilization analysis indicates that additional memory is required, the issue is now in the system configuration area. However, it could also imply a problem with the application being executed requiring excessive memory, in which case it becomes an application tuning issue. In reality, such an issue would probably be resource utilization, system configuration, and application tuning and would be solved as a combination. The four areas of system performance discussed herein provide an actual framework for the System Administrator to categorize system performance issues and to begin to investigate and address them.

The investigation and analysis of system performance issues requires one to retain a solid basic knowledge level of the system operation. Generally, the more experience and internal level knowledge that one has, the more successful one will be investigating, determining, and solving system performance problems. Unfortunately, much of the system performance experience necessary to be successful is available only by going through the experience itself. System performance issues can vary so widely between different systems, applications, etc. that no two performance issues are alike and therefore no two experiences are alike, even though parallels may often be drawn. One must continually be exposed to system performance to be successful investigating system performance. This means collecting performance data and examining that data on a regular basis. This means watching users perform tasks on the system and becoming familiar with their specific performance issues. This means constantly being in touch with how the system is performing.

SAR (System Activity Reporter) is an AT&T System V Interface Definition performance tool. It is designed to address performance issues within HP-UX systems specifically in the area of resource utilization, but also within system configurations. Basic performance parameters are obtained directly from the kernel counters and are made available for both logging to data files and reporting on a global level. SAR provides reporting functions to examine CPU utilization, swap information, system calls, I/O information, file accesses, and interprocess communication. Because SAR data is reported in a fairly raw format, interpretation is sometimes difficult.

SAR provides a basic data source for the System Administrator to access performance information maintained within the HP-UX kernel. This performance data is either logged to a binary file by SAR or immediately provided to the user as a report. If the performance data has been previously logged to a binary file, SAR can be used to generate performance reports using either the entire set of data logged or a subset of the data broken down by start and ending time or by the logging interval. The System Administrator can then examine these reports, interpret them, and recommend a possible solution to whatever performance difficulties are being investigated.

Typically, system performance problems can be broken down into both short term and long term problems. Short term problems would include those where the system performance has degraded quite suddenly, thus forcing the System Administrator to immediately begin investigating and proposing a solution to the problem. Contrast this with long term performance problems, such as a slowly increasing level of CPU activity. Long term problems would only become evident after examining and reviewing system performance data over a longer time period and identifying trends within that system performance data. SAR can be used to address both short and long term performance problems, provided that the System Administrator has a familiarity with SAR, its implementation and limitations.

As previously mentioned, the first step to utilizing SAR is to begin collecting data. The man pages SAR (1) and SA1 (1M), in Appendix A, provide details regarding the invocation and options necessary to run SAR. While complete, the man pages do fall short of providing specific examples for collecting and interpreting both short term and long term data.

Short term data can be collected, logged into a binary file, and displayed on the user's tty by using the following SAR command:

```
sar -A -o file t n
```

where -A indicates that all data reporting options should be reported, -o specifies the file name, t indicates the number of seconds between logging records, and n specifies the number of records to be logged or the number of intervals of t seconds. Generally, the -A option should be used so that all data is logged into the file. This allows the System Administrator to report and further restrict the data during the reporting process. The t option should generally be as small as possible such that enough detail is contained within the binary file for further analysis later.

Another mechanism for executing the same sar command presented above, but logging only to the binary data file, not the tty, is:

```
/usr/lib/sa/sadc t n file
```

which will invoke sadc, the system activity data collector, directly. A very useful example of short term data collection using the commands just discussed is:

```
/usr/lib/sa/sadc 60 20 temp &
```

which will provide logging of all system activities available to a binary file called temp in the current working directory at an interval of one minute for twenty minutes. This command would generally be invoked by a System Administrator who was examining a performance problem that was of a short term nature. Because this data is short term in nature, a minimal t second interval of 60 seconds or less is recommended. In this case the disc space and overhead associated with execution of the sar logging function is genr- .lly not an issue. The temporary file generated for data storage by sar will be reported and interpreted later. As such, the file used is temporary storage and can be removed after interpretation.

Long term data is collected by using the sa1 command. The invocation synopsis for sa1 is:

```
/usr/lib/sa/sa1 t n
```

where t is seconds between logging binary records to the file and n is the number of records to be logged. Please note that the sa1 command does not allow specification of a temporary data file. The file used defaults to /usr/adm/sa/sadd where dd is the current day. Thus anytime the sa1 command is invoked, it will log to this same daily data file. For this reason it is recommended that the sa1 command be set up in the /etc/rc script to be invoked whenever the system is rebooted. In this case the value of n should be made quite large to ensure that the records are continually logged and sar is not stopped. This mechanism will work well if the system is rebooted at least once per day. Cron can also be used as a scheduling mechanism.

When the system is rebooted, the counters within the sar reporting facility must be reset. When long term data is logged, there is a mechanism to specify a "reboot has occurred" record within the sadd binary data file. The entry necessary to write this binary record must be inserted into the /etc/rc script and is as follows:

```
/usr/lib/sa/sadc /user/adm/sa/sa'date +%d'
```

This entry is very important to allow the reporting function to differentiate the system reboot. If it is not executed at the time of system reboot, erroneous data within the sar reporting function will result.

Sa1 can also be time scheduled using the cron and crontab facility to log system performance information at varying times during the day. An excellent example of the crontab entries necessary to perform this function is presented in the SA1 (1M) man page. This example can be modified to suit any specific system installation.

When analyzing long term performance data it is important to first establish a performance baseline to indicate what "normal" performance is on the system. Because system configurations and applications can vary so widely, it is nearly impossible to examine performance data from one reporting instance and to draw any valid conclusions from it. It is therefore recommended that the long term sa1 data collection facility be set up and invoked at regular intervals on any system where performance is currently or may possibly become an issue. Only by establishing this performance baseline can the System Administrator hope to examine a later sar report taken when system performance is an issue, and to know what is normal performance and what is abnormal performance for the various system performance parameters reported by sar. Performance analysis is, in general, a skill learned through experience and experimentation. The importance of establishing a performance baseline for a specific system cannot be stressed enough.

There are four methods for generating system activity reports using sar. First, immediate reporting; second, reporting from a binary data file; thirdly, automatic reporting; and lastly, direct access from a user written c program.

System activity reports can be generated immediately on line by invoking sar with the following:

```
sar -options t n
```

which will immediately begin sampling the kernel parameters and reporting the system activity directly to the tty. In this instance it is highly recommended that sar be invoked with a minimum of options to narrow down and minimize the size of the printout being obtained. If this is not done the numbers generated will literally overwrite each other and be unreadable. A good example of reporting only the CPU utilization report is:

```
sar -u 60 10
```

which will report the CPU utilization once per minute for ten minutes. The -u option is probably the most used option in this immediate reporting mode.

System activity can also be reported from a previously logged binary data file. To do this the System Administrator would invoke sar as:

```
sar -A -f file
```

to generate a system activity report of all performance parameters for the entire data file. The activity report can be further restricted by using the -s and -e options. These options allow the user to specify a starting and ending time for the activity report thus narrowing the data within the file to a certain time period. A larger time interval can also be specified using the -i option. This allows the data within the data file to be further summarized by specifying a larger time interval then was specified when the data file was created. Thus, if the data file was created with a time interval of fifteen seconds, the activity report could be generated with a time interval of fifteen seconds, thirty seconds, forty five seconds, one minute, etc. Using the -i option, the System Administrator does not have to wade through large amounts of data to recognize performance trends within the system. Activity parameter reporting can also be restricted by specifying the specific data reporting options as noted within the SAR (1) man page. The -A option as used above specifies that all performance data is to be reported. The -s, -e, and -i, options are valid and usable for any of these reporting functions except the immediate reporting mode.

The third method of generating system activity reports is automatic reporting. This mode is invoked using the sa2 command as follows:

```
/usr/lib/sa/sa2 -A
```

to generate an activity report of all activities for the current day. As with the previous reporting option, the -s, -e, and -i, options can also be specified. The sa2 command assumes that the binary data file being used is /usr/adm/sa/sadd where dd is the current day. The system activity report is written to the file /usr/adm/sa/sardd where dd is the current day. This file can then be lp'ed or more'ed directly to the tty. The sa2 command can also be setup within cron to perform this reporting function automatically. An example of this is presented in the SA1 (1M) man page. An undocumented feature of the sa2 command is that it removes all data files and all reporting files under the /usr/adm/sa directory which are greater than seven days old. While this feature does keep the data files and reporting files from consuming large amounts of disc space on the system, it can be distressing to a System Administrator who is trying to set a performance baseline. Therefore the system administrator can either make sure that all reports and data files they wish to keep are saved elsewhere, or modify the sa2 shell script to not perform the file remove on the sa data files and reporting files. The sa2 script can be found in the /user/lib/sa directory.

The fourth mode of providing system activity reports is a user written c program. The binary data file format generated by sar is documented in the sa1 (1M) man page. This data structure can be incorporated by the c programmer directly into a c program for reporting purposes where the sar program reporting functions are not sufficient.

Once the system performance data has been collected, filed, and reported by sar, the next step is to interpret the data. This data interpretation will now be discussed case by case utilizing the data reporting options specified within the sar command.

CPU Utilization (-u).

These are the percentages of CPU time spent executing user processes, system mode, waiting for blocked I/O, and idle time. These percentages are calculated by examining the time spent by the CPU in these various states one-hundred times every second. At this time interval, the CPU state is examined and noted in the appropriate category. Waiting for blocked I/O includes:

1. The buffer cache block requested is already in use.
2. The buffer cache block requested already has I/O in progress.
3. Swapped or paged out pages are currently in the I/O state from the pageout daemon.
4. Raw I/O is being performed.

Waiting for blocked I/O means that no process could be run within the CPU because one of these four conditions existed. Generally, CPU utilizations greater than 85 percent should be considered as a maximum utilized CPU. A certain amount of data granularity will exist due to the algorithm used to report CPU utilization. This granularity can be reduced by looking at long term CPU utilization trends, generally with a greater than five to ten minute reporting interval. The average percentages are also reported for all data.

Buffer Cache Activity (-b).

This system activity report section helps to determine the efficiency of the system buffer cache. It reports both physical (bread, bwrit) and logical (lread, lwrit) of the system buffer cache buffers. The cache hit ratio is also calculated for both reads (%rcache) and writes (%wcache). This cache hit ratio should generally be in the vicinity of 85-90 percent for reads and somewhat less for writes. A definite problem exists within the system if both reads and writes fall well below the 75 percent mark. Again, these cache hit ratios should be compared to the system baseline performance discussed earlier. This section of the report also specifies the raw physical I/O to disk (pread, pwrite). All activity specified within this section are specified as operations per second, except for cash hit ratios which are a percentage. The average values for the data selected are also reported.

Block Device Activity (-d).
Block devices, such as the disk and tape drives, are reported with the percentage of time they are busy, the number and average length of data transfers, the average waiting time for service and average service times. The percent busy algorithm within the kernel examines each block device one-hundred times per second and checks to see if the device is busy. If the device is busy, it is noted within a counter and the number of requests waiting for that device is also noted. These counters are then averaged to provide the percent busy and avque report. The number of data transfers (r+w) and average size of data transfers (blks) are noted within the kernel by the block device I/O driver entries performed for each device. The average wait and service times are calculated from the time that the request has entered the queue until the time that the request has completed and the I/O is done. Block devices, especially disks, should be examined within this report for workload balancing using the percent busy parameter as an indicator. The average values for all block devices are also reported.

TTY Device Activity (-y).
This section of the report specifies the number of characters received on the raw tty input queue, the number of characters transmitted on the raw tty output queue, and the number of characters transferred from the raw tty input queue to the canonical input queue. In addition, the number of characters received from all multiplexer cards, and the number of characters passed to all multiplexer cards, along with the card's interrupt rates are reported. All of these items are reported in transfers per second. The averages are also reported.

System Calls (-c).
When the kernel traps any system call, trap.c is called to perform system call preparation. At this point, the fact that a system call has been performed is noted, and translated into a total for all types of system calls (scall). Other specific system calls that are reported include the system calls read and readv (sread) and write, writev calls (swrit). Forks and execs are also reported. The forks total includes both forks and vforks, and is counted when the new process is actually created. System calls which result in I/O are also noted (rchar, wchar) which specify the number of bytes transferred in total by bread, breada, and bwrit system calls in bytes. All of these items are reported in operations per second, and the averages are also reported.

System Swapping and Switching (-w).
Whenever a swap in or swap out process operation is initiated by the swapper process (proc [0]), the swapping action is counted in either swpin or swpot. Whenever the pageout daemon process (proc [2]) is called upon to bring memory pages into or out of the swap area, the number of bytes transferred is noted in bswin or bswot, depending upon input or output direction. Additionally, each time a new process is taken off the runqueue and started the process switch count is incremented (pswch). This is the total of the process context switches occurring within the system. All of these items are specified in operations per second in the report. Average values for all these parameters are also reported.

File Access System (-a).
This section of the report provides some basic performance data regarding the file system. Namei is the routine which returns the final inode in the last operation of a pathname lookup for a file. If running NFS, this always occurs on the local system. The iget routine gets an inode and puts it into memory buffers. An iget is done for each part of a pathname, provided the inode desired does not already exist in the memory buffer. The iget routine is always executed in an NFS situation on the system where the file system is originally mounted. Dirblk specifies the number of directory block lookups that have been done on the system. All of these items are reported in operations per second. Average values are

reported for all parameters.

Queue Lengths (-q).
Once per second the CPU scheduling subroutine within the kernel examines all processes to see if they are runable and in memory. The processes are examined in an unordered manner. Those that are currently runable and loaded in memory are considered to be on the runqueue and are noted as being runable (runq-sz) and those which are runable and not loaded (swapped out) are noted as such (swpq-sz). The percent occupied parameters indicate the percentage of time there is at least one process in the runqueue, or at least one process that is swapped out but ready to run. The average queue lengths and percent occupied parameters are reported too.

Table Status (-v).
This section of the report specifies the current usage of four tables within the kernel-text, proc, inode, and file. Also noted are the number of times that the tables have overflowed. The overflow count is noncumulative.

IPC Activities (-m).
This section reports both IP messages and semaphore activity. The messages per second indicates the number of messages received via the IP messages facility. The SEMA per second indicates the number of semaphore operations performed both acquiring and releasing semaphores. This item should be roughly two times the number of semaphores active within the system. The average values are also reported.

What has been presented thus far are the mechanics and methods of interpretation for using SAR as a system performance analysis tool. There are limitations to using SAR. First, it can be difficult to interpret and visualize the data provided. This is mainly because SAR presents its output in a tabular form. The user must become skilled in spotting trends, and using the best filtering mechanisms within SAR to help evaluate the data. Generally, this skill is only acquired through experience. For the occasional user, it is sometimes difficult to decide the proper interval necessary to gather the data required to examine a specific system performance problem. On the more positive side, SAR is an AT&T System V standard utility.

For the System Administrator involved with system performance issues on a more occasional basis there are two other basic resource usage tools which are probably better suited in this regard. Monitor is an unsupported tool which has a fairly wide spread distribution. Although unsupported by HP, Monitor will provide the System Administrator with basic performance data necessary to do some performance tuning. Monitor does provide access to some kernel parameters and counters which SAR does not.

Another system resource usage tool recently introduced by HP is Laser Rx/UX. This is a medium to long-term performance analysis tool which allows the System Administrator to collect data directly from the kernel via the built in measurement interface. This measurement interface is designed to have minimal impact upon the operation of the kernel while the data is being collected. Once the performance data has been collected, it is analyzed on a PC workstation, running the Laser Rx/UX graphical analysis software under Microsoft Windows. Laser Rx/UX provides a much more suitable user graphical interface which is ideal for spotting and analyzing long-term trends with regard to system performance.

With practice and regular usage, the System Administrator can become quite successful in analyzing and interpreting SAR reports. This data can then be used to spot performance trends and identify performance difficulties within the system. Again, it is important for the System Administrator to establish a performance baseline before undertaking a performance analysis using SAR.

APPENDIX A: Man PAGES

NAME

sar - system activity reporter

SYNOPSIS

sar [**-ubdycwaqvmA**] [**-o** file] t [n]

sar [**-ubdycwaqvmA**] [**-s** time] [**-e** time] [**-i** sec] [**-f** file]

DESCRIPTION

In the first form above, the *sar* command samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the **-o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second form, with no sampling interval specified, *sar* extracts data from a previously recorded *file*, either the one specified by **-f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **-s** and **-e** *time* arguments of the form *hh*[:*mm*[:*ss*]] The **-i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

-u Report CPU utilization (the default):
%usr, %sys, %wio, %idle - portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.

-b Report buffer activity:
bread/s, bwrit/s - transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s - accesses of system buffers;
%rcache, %wcache - cache hit ratios, e.g., 1 - bread/lread;
pread/s, pwrit/s - transfers via raw (physical) device mechanism.

-d Report activity for each block device, e.g., disk or tape drive:
%busy, avque - portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s - number of data transfers from or to device, number of bytes transferred in 512-byte units;

avwait, avserv - average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).

-y Report TTY device activity:

rawch/s, canch/s, outch/s - input character rate, input character rate processed by canon, output character rate;

rcvin/s, xmtin/s, mdmin/s - receive, transmit and modem interrupt rates.

-c Report system calls:

scall/s - system calls of all types;

sread/s, swrit/s, fork/s, exec/s - specific system calls;

rchar/s, wchar/s - characters transferred by read and write system calls.

-w Report system swapping and switching activity:

swpin/s, swpot/s, bswin/s, bswot/s - number of transfers and number of 512-byte units transferred for swapins (including initial loading of some programs) and swapouts;

pswch/s - process switches.

-a Report use of file access system routines:

iget/s, namei/s, dirblk/s.

-q Report average queue length while occupied, and % of time occupied:

runq-sz, %runocc - run queue of processes in memory and runnable;

swpq-sz, %swpocc - swap queue of processes swapped out but ready to run.

-v Report status of text, process, inode and file tables:

text-sz, proc-sz, inod-sz, file-sz - entries/size for each table, evaluated once at sampling point;

text-ov, proc-ov, inod-ov, file-ov - overflows occurring between sampling points.

-m Report message and semaphore activities:

msg/s, sema/s - primitives per second.

-A Report all data. Equivalent to **-udqbwcayvm**.

EXAMPLES

To see today's CPU activity so far:

```
     sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
     sar -o temp 60 10
```

To later review disk and tape activity from that period:

```
     sar -d -f temp
```

FILES

/usr/adm/sa/sa_dd_ daily data file, where _dd_ are digits representing the day of the month.

SEE ALSO

sa1(1M).

STANDARDS CONFORMANCE

sar: SVID2

NAME

sa1, sa2, sadc - system activity report package

SYNOPSIS

/usr/lib/sa/sa1 [*t* *n*]

/usr/lib/sa/sa2 [**-ubdycwaqvmA**] [**-s** *time*] [**-e** *time*] [**-i** *sec*]

/usr/lib/sa/sadc [*t* *n*] [*ofile*]

DESCRIPTION

System activity data can be accessed at the special request of a user (see *sar*(1)) and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include CPU utilization counters, buffer usage counters, disk and tape I/O activity counters, TTY device activity counters, switching and system-call counters, file-access counters, queue activity counters, and counters for inter-process communications.

Sadc and shell procedures, *sa1* and *sa2*, are used to sample, save, and process this data.

Sadc, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time to mark the time at which the counters restart from zero. The **/etc/rc** entry:

```
/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`
```

writes the special record to the daily data file to mark the system restart.

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in **crontab** (see *cron*(1M)):

```
0 * * * 0,6  /usr/lib/sa/sa1
0 8-17 * * 1-5  /usr/lib/sa/sa1 1200 3
0 18-7 * * 1-5  /usr/lib/sa/sa1
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar*, writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The crontab entry:

```
5 18 * * 1-5  /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 3600
-A
```

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
   struct sysinfo si;  /* see /usr/include/sys/sysinfo.h */
   int  sztext;        /* current entries of text table  */
   int  szinode;       /* current entries of inode table  */
   int  szfile;        /* current entries of file table  */
   int  szproc;        /* current entries of proc table  */
   int  msztext;       /* size of text table  */
   int  mszinode;      /* size of inode table  */
   int  mszfile;       /* size of file table  */
   int  mszproc;       /* size of proc table  */
   long  textovf;      /* cumul. overflows of text table  */
   long  inodeovf;     /* cumul. overflows of inode table  */
   long  fileovf;      /* cumul. overflows of file table  */
   long  procovf;      /* cumul. overflows of proc table  */
   time_t  ts;         /* time stamp, seconds  */
   long  devio[NDEVS][4];  /* device info for up to NDEVS units *
#define IO_OPS          0   /* cumul. I/O requests  */
#define IO_BCNT         1   /* cumul. blocks transferred */
#define IO_ACT          2   /* cumul. drive busy time in ticks  */
#define IO_RESP         3   /* cumul. I/O resp time in ticks  */
};
```

FILES

/tmp/sa.adrfl	address file
/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file

SEE ALSO

cron(1M), sar(1), timex(1).

STANDARDS CONFORMANCE

sa1: SVID2

sa2: SVID2

sadc: SVID2

APPENDIX B: SAMPLE SAR REPORT

```
HP-UX hpuspma A.B7.00 U 9000/835    04/27/90

08:34:22      %usr     %sys     %wio    %idle
08:36:22         2        6        0       92
08:38:23         2        7        0       91
08:40:23        42       58        0        0
08:42:31        47       53        0        0
08:44:22        50       50        0        0
08:46:22        47       53        0        0

Average         31       37        0       32

08:34:22   device    %busy    avque    r+w/s   blks/s   avwait   avserv

08:36:22 disc0-2         2      1.6        1        2     14.8     26.3
         disc2-0         3      1.0        1        7      0.0     30.1
         disc2-1         1      1.0        0        1      0.0     31.7

08:38:23 disc0-2         6      4.5        2        7     88.8     25.6
         disc2-0         3      1.0        1        6      0.0     31.0
         disc2-1         0      1.0        0        1      0.0     33.1

08:40:23 disc0-2        25     12.0       11       40    253.9     23.1
         disc2-0        58      1.0       18      133      0.0     32.2
         disc2-1        14      1.0        5       41      0.0     24.8

08:42:31 disc0-2        25      9.7       10       35    227.4     26.0
         disc2-0        92      1.0       30      175      0.0     30.5
         disc2-1         9      1.0        3       21      0.0     27.2

08:44:22 disc0-2        35      8.5       13       47    196.9     26.3
         disc2-0        88      1.0       30      184      0.0     29.3
         disc2-1         5      1.0        2       13      0.0     29.6

08:46:22 disc0-2        31     10.4       12       42    247.3     26.2
         disc2-0        92      1.0       33      175      0.0     28.2
         disc2-1         5      1.0        2       12      0.0     32.5

Average  disc0-2        20      9.6        8       28    218.1     25.5
         disc2-0        54      1.0       18      110      0.0     29.8
         disc2-1         6      1.0        2       15      0.0     27.3

08:34:22 runq-sz %runocc swpq-sz %swpocc
08:36:22     1.1      15
08:38:23     3.1      14
08:40:23    29.9     107     1.5      12
08:42:31    42.1     112     9.7     108
08:44:22    41.6     112    20.5     112
08:46:22    50.0     113    23.8     113

Average     38.3      77    17.5      55

08:34:22 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
08:36:22       0     149     100       2       3      44       0       0
08:38:23       1      20      93       2       6      71       0       0
08:40:23       3     268      99      17     178      90       0       0
08:42:31       3     214      98      15     160      91       0       0
08:44:22       5     193      97      13     146      91       0       0
08:46:22       4     225      98      11     124      91       0       0

Average        3     175      98      10     100      90       0       0

08:34:22 swpin/s bswin/s swpot/s bswot/s pswch/s
08:36:22    0.01     0.3    0.00     0.0      14
08:38:23    0.23     7.1    0.00     0.0      16
```

```
08:40:23    1.56    34.9   11.99   385.7     144
08:42:31    5.30    76.6   14.83   460.5     341
08:44:22    7.72    94.0   12.83   549.4     332
08:46:22   12.40   180.9   11.81   452.5     488

Average     4.34    63.0    8.29   297.3     214

08:34:22 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
08:36:22     176     117       4    0.11    0.11 1194095    9649
08:38:23     152      40       5    0.48    0.44   71391    8334
08:40:23    1132     336      75    2.26    2.07 1258903   98788
08:42:31     962     329      65    1.38    1.23 1152268   86292
08:44:22     908     319      63    1.01    0.92 1046548   66618
08:46:22     811     308      58    0.87    0.76 1340149   52624

Average      674     236      44    1.00    0.91  997931   52424

08:34:22  iget/s namei/s dirbk/s
08:36:22       1       5       1
08:38:23       8      42       4
08:40:23      80     412      16
08:42:31      50     250       9
08:44:22      48     218       7
08:46:22      47     170       7

Average       38     179       7

08:34:22 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
08:36:22       1       0     188       0       0       0
08:38:23       1       0     145       0       0       0
08:40:23       0       0     292       0       0       0
08:42:31       0       0     229       0       0       0
08:44:22       0       0     233       0       0       0
08:46:22       0       0     182       0       0       0

Average        0       0     210       0       0       0

08:34:22 text-sz  ov proc-sz  ov inod-sz  ov file-sz ov
08:36:22  50/108   0  89/660   0 771/772   0 223/1256  0
08:38:23  53/108   0 110/660   0 766/772   0 291/1256  0
08:40:23  54/108   0 226/660   0 771/772   0 618/1256  0
08:42:31  54/108   0 298/660   0 767/772   0 811/1256  0
08:44:22  55/108   0 340/660   0 768/772   0 949/1256  0
08:46:22  53/108   0 382/660   0 765/772   0 1071/1256  0


08:34:22   msg/s  sema/s
08:36:22    1.01    0.00
08:38:23    1.00    0.00
08:40:23    1.04    0.00
08:42:31    1.10    0.00
08:44:22    1.09    0.00
08:46:22    1.09    0.00

Average     1.05    0.00
```

TITLE: The HP 1000 A990 - The Time Has Come!

Author: Rita Sandhu

AUTHOR: For more information contact:

Ella Washington

Hewlett-Packard

19091 Pruneridge Ave. MS 46LK

Cupertino, CA 95014 408-447-1053

FINAL PAPER WAS NOT AVAILABLE AT TIME OF PRINTING

PAPER NO. 1082

Introducing the New HP 1000 SCSI Interface Card.

Alan Tibbetts
Consultant
Hewlett Packard Co.

This session will provide an overview of the SCSI interface card for the HP 1000 A-Series computers. A short overview SCSI peripherals will be presented before attendees will be taken on a quided tour of the block diagram of the card. The unique features of the backplane interface will be explained along with the driver to card communications method.

The concept of 'scripts' will be introduced and examples will be presented that show how the HP drivers use them to perform data transfers using the card.

SPOOLING AS PART OF ADVANCED SYSTEM MANAGEMENT

Martin Kellner
HEWLETT-PACKARD GMBH
Herenberger Strasse 130
D-7030 Boeblingen
West-Germany

Introduction

Hewlett-Packard Company is happy and proud to announce today a new spooling solution for its entire HP-UX platform, comprising all computers of the HP 9000 Series 300/600/800: The product, called HP OpenSpool/UX, is designed for networked and stand-alone environments, addressing both commercial and technical applications. Besides a full set of capabilities introducing unmatched spooling functionality to the Open Systems market, HP OpenSpool/UX reflects a new level of System Management characteristics as demanded in cooperative computing environments.

Before exploring some of the key contributions and benefits provided with this new spooling concept, let us have a quick review of the driving forces behind System Management. What is it that creates this new momentum? Which factors fuel a growing importance for smart System Management? In short, two phenomena seem to lead this development: emerging technology and the human being.

FIGURE 1

Why System Management?

Proliferating networks at different department levels, customers' demand for multi-vendor interoperability, and the newly evolving client/server architecture contribute to more complex environments ("systems") that still need to be managed.

On the other hand, the human being is continually changing behavior and expectations. This includes the request for fewer training investments. A French user, for example, doesn't want to learn English to use his or her workstation. While striving for higher productivity, the human being is prepared for a new role. Users are willing to take over new responsibilities as long as they feel supported by their computing resources.

"System Management" is used to describe the area within the computer market that tackles this new opportunity in the form of new tools and solutions. HP closely follows X/Open in the definition of application areas and components that make up System Management.

Figure 2

Target Environment

Before focusing further on Output Management or spooling solutions, the targetenvironment for HP's System Management activities must be defined. This could
be a stand-alone multiuser system as shown in figure 3, or a client/server environment, consisting of a couple of workstations and servers connected to a LAN, as shown in figure 4. In fact, any combination of interconnected Series 300/600/800 systems via LANs is provided for in our strategy, leaving it to the customer to integrate different departmental solutions in the future.

FIGURE 3

FIGURE 4

The Spooler Opportunity

Facing the various components of System Management, the decision to invest in which application areas first is certainly influenced by market opportunities. In the Open Systems market, spooling clearly represents one of these opportunities.

From the user's point of view, limited network capabilities, missing transparency, and a low level of user comfort, must be observed. Conflicting local-only spooling implementations, numerous customized work-around solutions, and an overall limited support of available output devices contribute to a diminishing customer satisfaction.

From the System Administrator's point of view, isolated, incompatible spooling solutions represent a serious problem on its own with an accumulating trend due to a growing interconnection of individual implementations. System Management becomes a true challenge.

FIGURE 5

With this in mind, the idea was born at HP to launch the development of a new spooling concept providing a single, network-wide spooling solution across any possible LAN-based HP-UX topologies. The result is HP OpenSpool/UX, designed for novice and experienced users in technical and commercial environments.

True Distributed Spooling

Users of HP OpenSpool/UX can trace their print or plot requests until final execution for last-minute corrections. End-users need not worry about the actual number of servers involved. As shown in figure 6, the truly distributed-spooling concept of HP OpenSpool/UX provides customers with effective, network-wide utilization of printers and plotters, while hiding network complexity.

FIGURE 6

Outstanding Functionality

HP OpenSpool/UX serves technical and commercial applications equally well. For commercial environments, this implies extensive forms and fonts handling to exploit the latest features of ever smarter output devices. The spooler is prepared for fast changing print requests (paper size, number of copies, character sets, preprinted forms), high-volume printing, and multiple security levels as requested when printing checks.

In technical environments, a single spooling solution for prints AND plots will replace numerous existing "work-arounds." Viewing plot requests across networks to change priorities or modify further attributes, ensures maximum flexibility. The saving of print or plot requests after execution is also possible. Repetitive tasks are facilitated through user-definable templates. HP OpenSpool/UX ensures that the user benefits from the latest features of all output peripherals currently supported on HP-UX.

FIGURE 7

Tailored User Interface

To better accommodate the individuals' level of expertise, HP OpenSpool/UX provides the choice of a commands- or menu-driven user interface. In addition, context-sensitive, on-line help facilities guide the user through the various tasks effectively and smoothly, ensuring short learning curves. Those users working exclusively through menus and templates will not be overwhelmed by command details. They will not even realize the underlying HP-UX operating system, while HP-UX experts may not want to give up the efficiency provided at the command level.

Advanced System Management Characteristics

With HP OpenSpool/UX, the evolving importance of network-wide System Management is recognized in various forms. Single-point administration provides for transparent spooling management from any console in the network (LAN). A rich set of default parameters encourages and simplifies initial installation by the customer. Facilities such as one-hour-up installation and the choice of a centralized versus distributed spooling management, are part of this new concept to better address today's user needs.

FIGURE 8

The (optional) new role of different user types is considered by a fine split of the possible tasks to be addressed for smooth, day-to-day spooling operation. This scalable end-user involvement is detailed in Figure 9. However, the listed tasks can still be handled all by a single person, the administrator.

Figure 9

Flexible Integration

The emulation of the X/Open defined commands "lp," "lpstat," and "cancel" supports a smooth integration of HP OpenSpool/UX with customers' existing applications. In the same way, the use of customized filters and fonts is supported. An increasing request for PC integration via LAN Manager/X is also perfectly complemented with HP OpenSpool/UX.

Built-In Future

The product's modular architecture ensures adaptation to a wide range of different HP-UX topologies, including diskless clusters. The individual software components, as shown in Figure 10, can either reside on different client or server systems, or on a single stand-alone machine. Network Computing Services (NCS) from the HP/Apollo division was selected for the implementation of the necessary Remote Procedure Calls (RPCs). The core spooling functionality is based on a joint development between Digital Equipment Corporation, the Massachusetts Institute of Technology (MIT), and Hewlett-Packard Company. This vendor-independent spooler communication layer presents the groundwork for cooperative computing in multi-vendor environments. The compliance with standards (X/Open, ECMA, ISO) at various levels anticipates future trends: increasing number of Open Systems, higher degree of networking, demand for better interoperability, and improved System Management.

FIGURE 10

Outlook

Looking into the future, there is still a wide area to be addressed in order to provide a single, consistent spooling solution for truly cooperative (multi-vendor) computing environments. Currently, HP is investigating in two directions: portability and interoperability. Part of this phased approach is the first release of HP OpenSpool/UX, offering a single HP-UX wide spooling solution. Portability will include new revisions to keep pace with newly available output devices from HP, as well as extensions to address additional operating systems.

Where portability does not provide an appropriate solution, a smooth inter- operability will lead HP's strategy. In this case, a common user interface and a comprehensive set of System Management capabilities are key objectives.

All of these activities must be seen in support of integrated System and Network Management as one key enabler of HP's company strategy "HP NewWave Computing" for cooperative multi-vendor computing, demanded by our customers.

Figure 11

Figure 1

System Management
The Driving Forces

Technology

- proliferating networks
- multi–vendor interoperability
- client/server architecture

Human belng

- shorter learning curves
- strive for higher productivity
- new role of the end–user

Computer Systems Boeblingen
IE005/MK/02.05.90

HEWLETT
PACKARD

Figure 2

SM Application Areas

File System Administration

Backup & Restore

Application Installation

User Accounting & Management

Batch Job Management

System Tuning & Reconfiguration

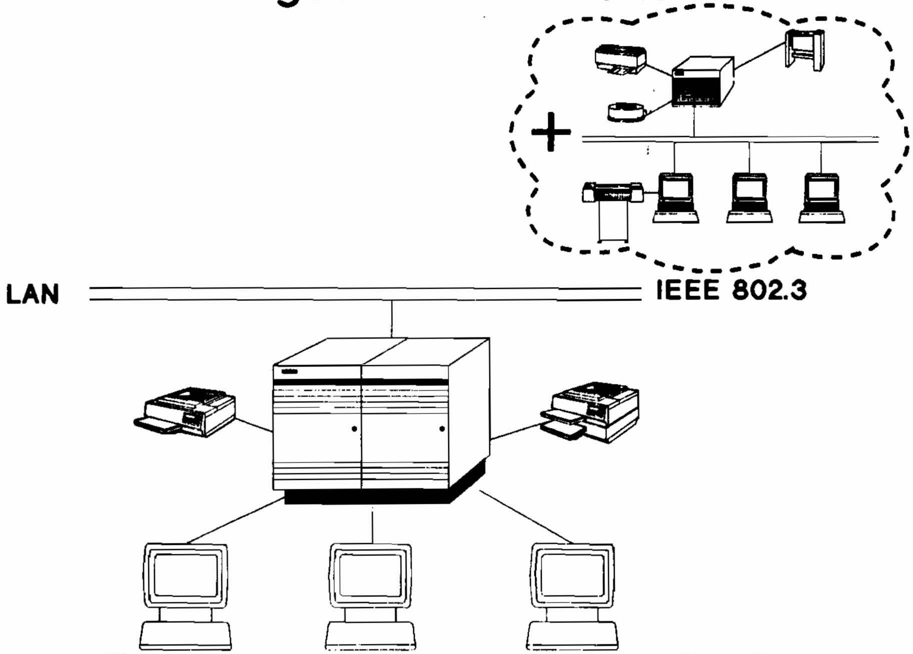
Performance Monitoring

Printout Management

Source: X/Open 1990 Technical Program
Systems Admin & Management

Figure 3

HP OpenSpool/UX
Target Environment

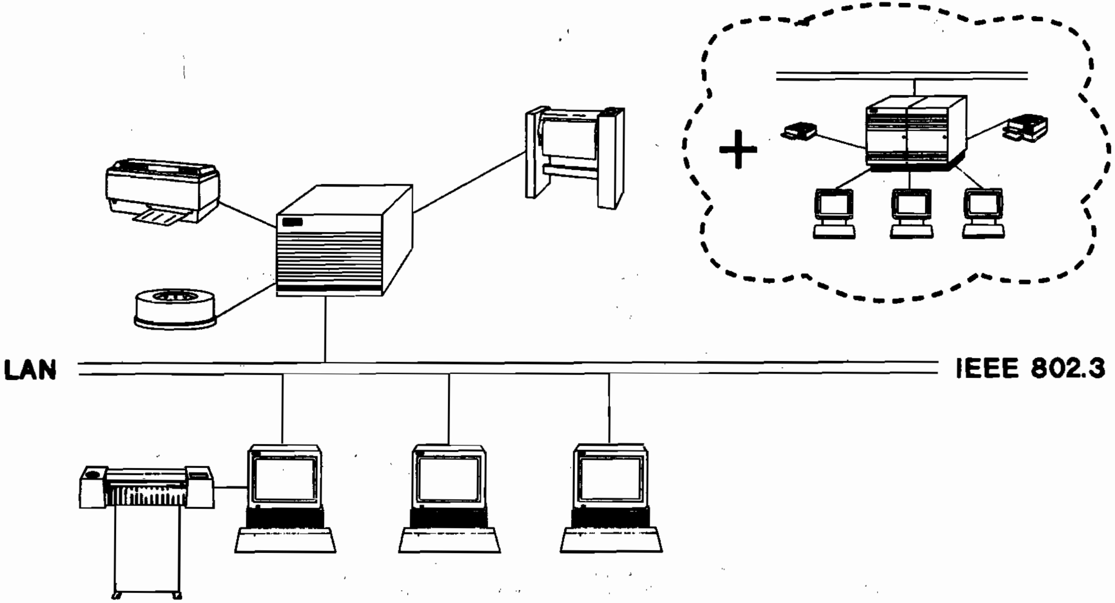


Computer Systems Boeblingen
MK005b/26.02./DK

HEWLETT PACKARD

Figure 4

HP OpenSpool/UX
Target Environment



Computer Systems Boeblingen
MK005a/26.02./DK

HEWLETT
PACKARD

Figure 5

THE SPOOLER OPPORTUNITY

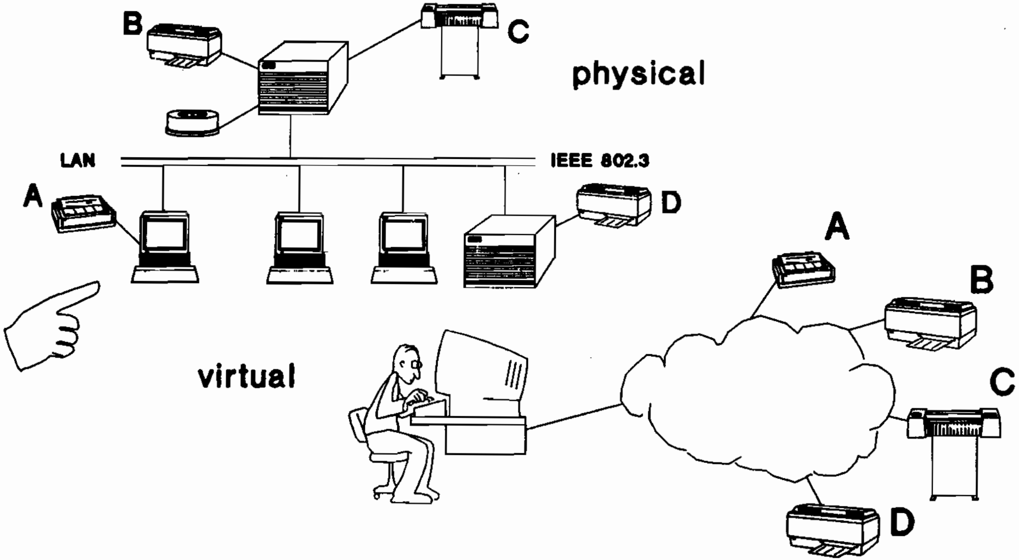
- □ Lacking network capabilities and user comfort
- □ Conflicting implementations of spoolers due to products of different origin:
  - few local 3rd party offerings
  - numerous customized work–around solutions (scripts)
- □ Limited support of available printer features, insufficient access to plotters
- □ Incompatible spooler systems while installation and updates demand a consistent spooler version across the network

Computer Systems Boeblingen
MK001/11.01./DK

HEWLETT
PACKARD

Figure 6

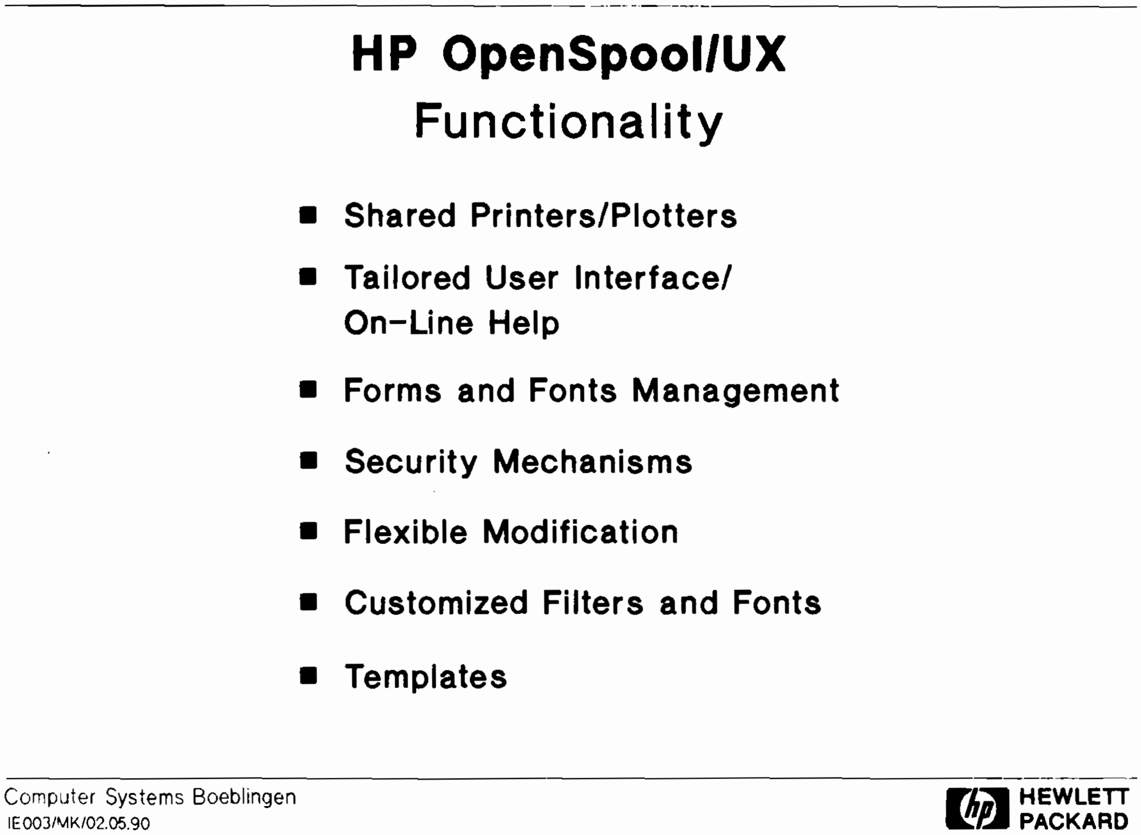
HP OpenSpool/UX
Hiding Network's Complexity



Computer Systems Boeblingen
MK021a/30.01./DK

HEWLETT PACKARD

Figure 7

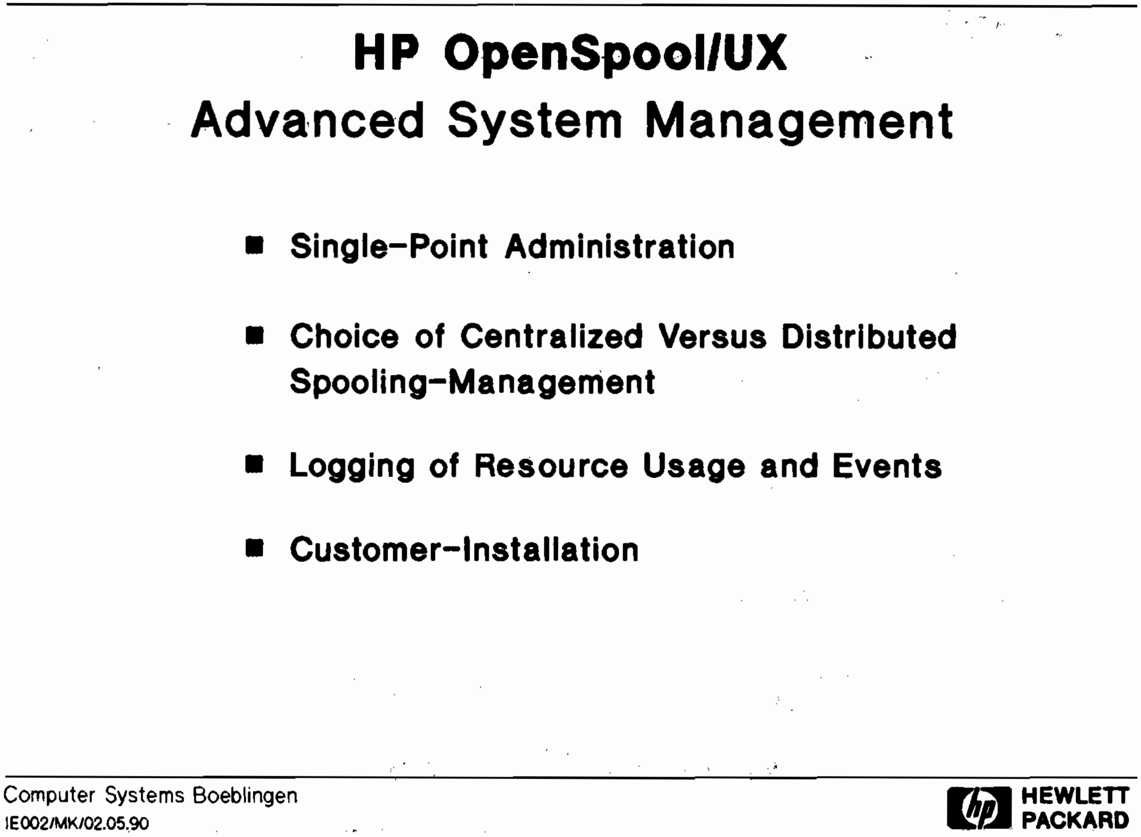


HP OpenSpool/UX
Functionality

- Shared Printers/Plotters
- Tailored User Interface/
  On-Line Help
- Forms and Fonts Management
- Security Mechanisms
- Flexible Modification
- Customized Filters and Fonts
- Templates

Computer Systems Boeblingen
IE003/MK/02.05.90

HEWLETT
PACKARD



HP OpenSpool/UX

Advanced System Management

- Single-Point Administration
- Choice of Centralized Versus Distributed Spooling-Management
- Logging of Resource Usage and Events
- Customer-Installation

Computer Systems Boeblingen
IE002/MK/02.05.90

HEWLETT PACKARD

Figure 8

Figure 9

HP OpenSpool/UX

Profile of the Actual User

- □ Spooler Administrator – installs S/W and maintains the spooler configuration
- □ Queue Manager *) – manages the print queues
- □ Device Manager *) – responsible for printers and plotters
- □ End–User – handles own spool requests if desired

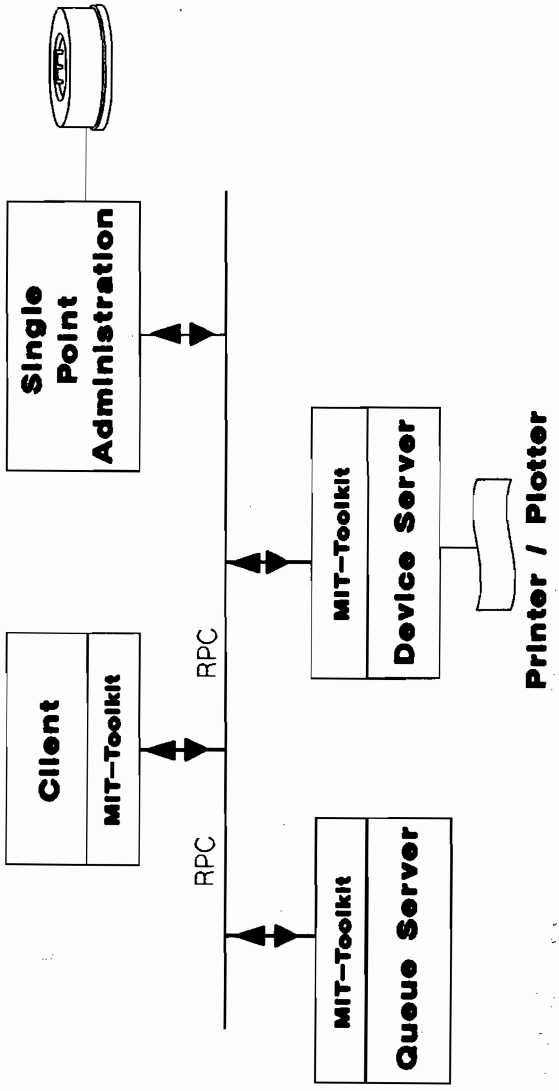
*) In an EDP environment, this is covered by the operator.

Computer Systems Boeblingen
MK011/09.01./DK

HEWLETT PACKARD

Figure 10

HP OpenSpool/UX
Modular Architecture



Computer Systems Boeblingen
MK009A/02.05.90

HEWLETT PACKARD

Figure 11

Outlook

Spooling in cooperative computing environments

- ■ Portability
  - – new releases
  - – further operating systems
- ■ Interoperability
  - – common user interface
  - – comprehensive System Management enabler: OSF distributed-computing functions

➡ The foundation of HP NewWave Computing:
Integrated System & Network
Management (HP OpenView)

Computer Systems Boeblingen
IE001/MK/02.05.90

HEWLETT
PACKARD

Multivendor Networking
with
NS-ARPA/1000

Reza Memar
Information Networks Division
Hewlett-Packard Co.
19420 Homestead Road
Cupertino, CA 95014

Introduction

NS-ARPA/1000 provides communication to other NS-ARPA/1000 nodes, with DS/1000-IV nodes using DS services, with NS nodes using NS services, and with other vendors using ARPA services. Since the introduction of NS/1000 there has been a misconception that NS/1000 does not have a true TCP/IP protocol stack. The primary reason for this was the absence of the Ethernet protocol at the data link layer and the lack of ARPA services (TELNET, FTP), which prevented communication with other members of the Internet community. Therefore at first release, NS/1000 could only communicate with other HP machines that were supporting both common services (e.g. NFT) and common links (e.g. 802.3).

At the 5.16 release of NS-ARPA/1000 Ethernet and FTP were added. Since then, it has been possible to communicate with other vendors using ARPA services over defacto standard Ethernet links. At the 5.2 release of NS-ARPA/1000 IP Subnetting was added. Now you can have HP-1000s, PCs, and workstations, all on the same LAN. At the same time, since NS-ARPA/1000 provides backward compatibility with DS/1000-IV, you can communicate with older HP-1000 machines.

In the open networking environment of today, it is important to be able to communicate with other members of the network using standard protocols. NS-ARPA/1000 provides both industry standard protocols to communicate with other vendors' machines and HP proprietary protocols to communicate with older HP machines.

This paper focuses on the industry standard protocols and services that NS-ARPA/1000 provides. It explains the internal protocol layering of NS-ARPA/1000. It explain the basic functioning and usage of TELNET and FTP with emphasis on communication with non-HP1000 systems. Finally it explains how you can use NetIPC sockets to communicate with BSD Sockets on other machines. This paper will not discuss about NS or DS services.

I. Terms and Abbreviations

First let's explain the terms and abbreviations which are used in this paper as well as in industry.

- **802.3**
  A IEEE Local Area Network standard protocol. It uses CSMA/CD technology to deliver data on the physical link.

- **Ethernet** A popular Local Area Network Technology invented at the Xerox corporation Palo Alto Research Center. It uses CSMA/CD Technology to deliver data on the physical link.

- **Internet Protocol (IP)** The Internet standard Protocol that defines the internet datagram as the unit of information passed across the internet and provides the basis for the internet connectionless, best-effort packet delivery service.

- **Internet Control Message Protocol (ICMP)** An integral part of the Internet Protocol (IP) that handles error and control messages. Specifically, gateways and hosts use ICMP to send reports of problems about datagrams back to the original source that sent the datagram.

- **Address Resolution Protocol (ARP)** The Internet protocol used to dynamically bind a high level internet address to a low level physical hardware address. ARP is only across a single physical network and is limited to networks that support hardware broadcast (i.e. ethernet).

- **Probe** HP-proprietary protocol which has two functions. Similar to ARP, Probe dynamically binds an IP address to a low level physical address. In addition, Probe dynamically binds a node name to an IP address. Probe operates only across a single physical network and works on 802.3 LANs.

- **Transmission Control Protocol (TCP)** The internet standard transport level protocol that provides the reliable, full duplex, stream service on which many application protocols depend. TCP allows a process on one machine to send a stream of data to a process on another machine. It is connection-oriented in the sense that before transmitting data, participants must establish a connection. Internet protocol suite is often referred to as TCP/IP.

- **TELNET** The Internet standard protocol for remote terminal connection service. TELNET allows a user at one site to interact with a remote timesharing system at another site as if the user's terminal were connected directly to the remote machine. Usually implemented at the application level, TELNET uses TCP as the underlying transport.

- **File Transfer Protocol (FTP)** The Internet standard protocol for transferring files from one machine to another. Usually implemented at the application level, FTP uses TCP as the underlying transport.

- **Sockets** The abstraction provided by Berkeley UNIX that allows a process to access the transport. A process creates a socket and connects to a specific destination and then sends and receives data from that socket.

- **NetIPC Sockets** The abstraction provided by NS-ARPA/1000 that allows a process to access the transport. Like BSD Sockets, A process creates a socket and connects to a specific destination and then sends and receives data from that socket.

II. Internal Structure of NS-ARPA/1000

Figure 1 shows the main components of the NS-ARPA/1000 product . The main body of the product consists of the inbound (INPRO) and outbound (OUTPRO) protocol processes. On one side they communicate with the network links and on the other, they communicate with application programs. In an idle system, INPRO sleeps on its class number, waiting for the link layer interface to queue a message on its class. OUTPRO, on the other hand, is waits on a resource number in the idle state. Any process that needs OUTPRO's processing (i.e. sending data across the network), would queue the data/request on the appropriate socket and wake up OUTPRO by unlocking its resource number. All of the NS-ARPA/1000 transport protocols namely, TCP, IP, ARP, Probe, and ICMP are implemented in these two programs.

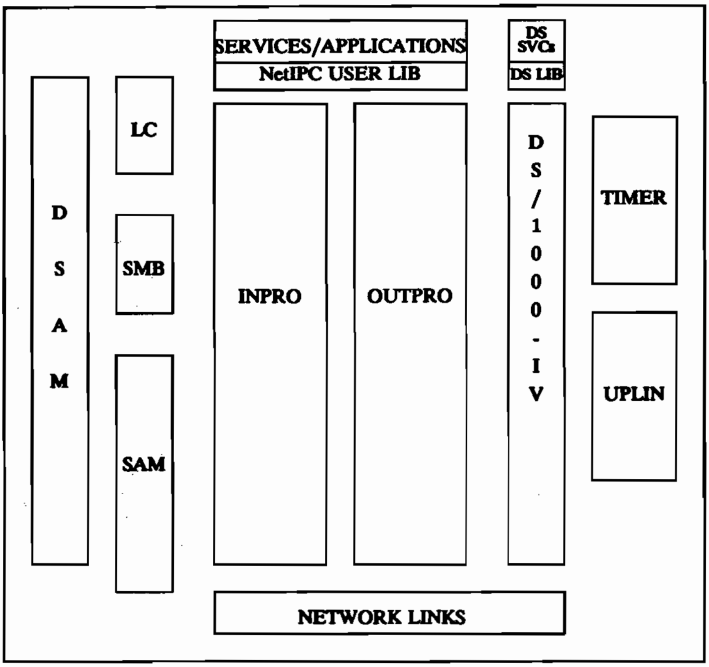


Figure 1: NS-ARPA/1000 Overview

NS-ARPA/1000 utilizes memory space in four different locations for its tables, message buffers, and global variables.

- DS System Available Memory (DSAM) is the area for tables, message buffers, and global variables relating to NS and ARPA services. There is a memory manager, similar to the Berkeley UNIX memory manager, which is responsible for allocation and maintenance of DSAM. Units of dynamic memory are in 128 bytes chunks (mbufs) and 1024 bytes chunks (clusters).

- Label Common (LC) and System Memory Block (SMB) are used by DS/1000-IV services and transport. LC contains global variables and SMB contains DS/1000-IV tables (some of the

DS/1000-IV tables are kept in DSAM as well).

- System Available Memory (SAM) is used to send and receive the data to and from communication drivers.

Data arriving over a network link will be placed in SAM by the communication driver. It is then copied to DSAM by the inbound process (INPRO). After processing by the protocol stack, the data is made available to the user program by the user library that copies the data from DSAM into the user buffer.

Outbound data out will be copied from the user buffer into DSAM by the user library routines. Then each protocol (i.e TCP and IP), adds its header information to the data. When the data is ready to go out onto the network, it is copied from DSAM to SAM where it is picked up by the communication driver for delivery. A copy of the message is kept in DSAM for possible retransmission.

In addition to INPRO and OUTPRO which provide the bulk of the processing, there are two processes which are essential to the proper functioning of the NS-ARPA/1000 product.

The TIMER program provides a facility to set various timers for different protocol modules. INPRO and OUTPRO communicate with TIMER program through class I/O.

The UPLIN Program is responsible for cleaning up after application program termination as well as NS-ARPA/1000 programs. It returns resources utilized by NS-ARPA/1000 programs to the system..

DS/1000-IV services and transport are the same as in DS/1000-IV product, except some of the DS tables have been moved to DSAM. Also there are some hooks both in the NS-ARPA protocol stack and DS/1000-IV transport which enable routing NS-ARPA packets over DS links and DS packets over NS-ARPA links. For more information consult the NS-ARPA/1000 Initialization Manual.

The relevant protocol layering of NS-ARPA/1000 is shown in figure 2. Although there are other protocols which are part of NS-ARPA/1000, they are not utilized in a multivendor communication. For example, Packet Exchange Protocol (PXP) is implemented on HP machines only and it is not used to communicate with Non-HP systems.

As illustrated, TCP/IP is the NS-ARPA/1000 transport and both Ethernet and 802.3 LAN links are offered. This combination makes the NS-ARPA/1000 product capable of communicating with other vendors who support TCP/IP protocols. The interface to the protocol stack is through NetIPC Sockets which provide functionality very similar to Berkeley Sockets (BSD Sockets) in the UNIX environment. In a later section we will discuss the similarity between NetIPC and BSD Sockets and we will show that they are only local data structures, rather than network protocols.

TELNET and FTP are the two industry standard services which are offered in this product. They both use NetIPC to access TCP ports and communicate with remote nodes. We will discuss these two services in later sections. Remote Program Management (RPM) and Network File Transfer (NFT) are HP proprietary services. Discussion of these two services is outside of the scope of this paper. For more information consult the NS-ARPA/1000 Users/Programmers manual.

This product gives you a choice at the link level. Depending on your need, you may want to enable your link as Ethernet or 802.3 or both. If you choose to enable your link as both Ethernet and 802.3, then on the same physical LAN you can send and receive both types of packets with the same interface card. For inbound traffic, the LAN driver will send both types of packets up to INPRO. For outbound traffic, the proper link type for communication is determined at connection establishment time by sending out a packet of each type to the remote node. The type of the reply packet is used by the local node for all future communication with the remote node on this connection. This provides the flexibility of communicating over both 802.3 and Ethernet on the same LAN.

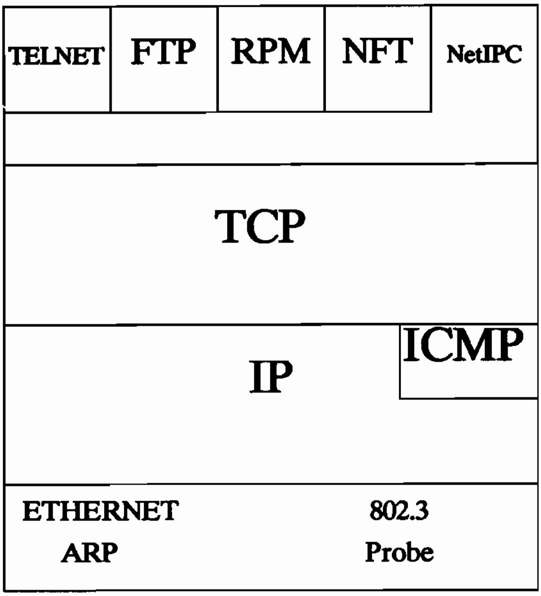


Figure 2: Protocol Layers in NS-ARPA/1000

III. FTP

FTP is an ARPA Service that allows you to transfer files among HP-1000 and other machines that support ARPA services. On an HP-1000 it consists of three programs:

FTP.RUN	FTP user program
FTPMN.RUN	FTP call monitor program
FTPSV.RUN	FTP server program

Figure 3 illustrates the interaction between these programs. A user starts an FTP session by running the FTP program:

FTP *remote-host-name* or *remote-IP-address*

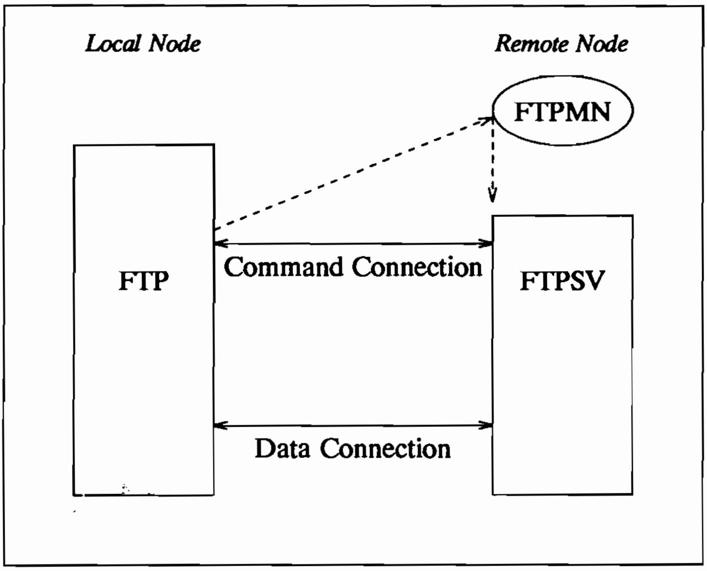


Figure 3: FTP Processes

The FTP program sends a request to the remote node to establish a connection to a well-known TCP port number (port 21). On the remote node, the call monitor program (FTPMN) is listening on TCP port 21. Upon receipt of the connection request, FTPMN completes the connection with the user program. Then FTPMN starts a copy of the server program (FTPSV), and gives the connection to the server program and goes back to listen to port 21 for other connection requests. FTPSV will ask the user for login and password. After verification of the login and password, FTPSV create a login session and attaches itself to that session. At this time the user may send commands through this connection to the server program. If a data transfer is to be performed, the FTP user program will send its own port address, which it expects to establish a connection on, to FTPSV via the command connection. Then, the server program sends a connection establishment request to that port number and a data connection will be established. When data transfer is done this connection is closed. The same sequence will be repeated if another data transfer is required. Although the above description is based on the HP-1000 implementation of FTP, most other implementations follow the same model.

A. FTP User Interface

User interface to FTP on the HP-1000 is very similar to the FTP interface on UNIX. The following is a list of FTP/1000 user commands:

!	Invokes CI on the local host.
?[?]	Displays FTP commands and help information. Same as HELP.
..	Sets the working directory on the remote host to the parent directory.
/	Displays the FTP command stack.
AP[PEND]	Transfers *local_file* to the end of *remote_file*.
AS[CII]	Sets the FTP file transfer type to ASCII. This is the default type.
BE[LL]	Sounds a bell after each file transfer completes.
BI[NARY]	Sets the FTP file transfer type to binary.
BY[E]	Closes the remote connection and exits from FTP. Same as EXIT and QUIT.
CD	Sets the working directory on the remote host to the specified *remote_directory*.
CL[OSE]	Closes the remote connection and remains in FTP.
DEB[UG]	Prints the commands that are sent to the remote host.
DEL[ETE]	Deletes the specified *remote_file* or empty *remote_directory*.
DI[R]	Writes an extended directory listing of a remote directory or file to the terminal or to a local file.
E[XIT]	Closes the remote connection and exits from FTP. Same as BYE and QUIT.
F[ORM]	Sets the FTP file transfer form to the specified format. The only supported format is non-print.
G[ET]	Transfers *remote_file* to *local_file*. Same as RECV.
GL[OB]	Toggles file name globbing.
HA[SH]	Toggles hash-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes.
HE[LP]	Displays FTP commands and help information. Same as ? and ??.
LC[D]	Sets or prints the local working directory.
LL	Specifies a log file to which FTP will send the commands and miscellaneous messages ordinarily displayed to the user's terminal.
LS	Writes an abbreviated directory listing of a remote directory or file to the terminal or to a *local_file*.
MDE[LETE]	Deletes multiple *remote_files*.

MDI[R] Write an extended directory listing of remote directories or files to a *local_file*.

MG[ET] Transfers multiple *remote_files* to the local system, using the same file names.

MK[DIR] Creates a *remote_directory*.

ML[S] Writes an abbreviated directory listing of remote directories or files to a *local_file*.

MO[DE] Sets the FTP file transfer mode to the specified mode. The only supported mode is stream.

MP[UT] Transfers multiple *local_files* to the remote system, using the same file names.

O[PEN] Establishes a connection to the remote host.

PR[OMPT] Toggles interactive prompting.

PU[T] Transfers *local_file* to *remote_file*. Same as SEND.

PW[D] Writes the name of the remote working directory to the terminal.

QUI[T] Closes the remote connections and exits FTP. Same as BYE and EXIT.

QUO[TE] Sends arbitrary FTP server commands to the remote host.

REC[V] Transfers *remote_file* to *local_file*. Same as GET.

REM[OTEHELP] Requests help information from the remote host.

REN[AME] Renames a *remote_file* or *remote_directory*.

RM[DIR] Deletes empty *remote_directory*.

SE[ND] Transfers *local_file* to *remote_file*. Same as PUT.

SENDP[ORT] Toggles the use of PORT command.

STA[TUS] Writes the current status of FTP to the terminal.

STR[UCT] Sets the FTP file transfer structure to the specified structure. The only supported structure is file.

TR **Specifies an input file from which to get FTP commands.**

TY[PE] Sets the FTP file transfer type to the specified type. ASCII and BINARY are the types currently supported.

U[SER] Logs into the remote host on the current connection, which must already be open.

V[ERBOSE] Toggles verbose output. When verbose output is enabled, FTP displays responses from the remote host.

Bold lines show commands which are offered by FTP/1000 which are not available on UNIX.. These are all local commands which give the HP-1000 user an RTE flavor. Using these commands you can setup a transfer file to run FTP in the background; you can also log the output of the FTP session to a file. The RTE command stack feature is also provided for easy interactive use.

B. Supported Features of FTP

The main objective in designing FTP was to achieve UNIX connectivity. The idea was that FTP is an industry standard protocol and UNIX implementation is the defacto standard implementation.

Supported features of FTP/1000 closely match UNIX implementation:

FORM	The only supported format is non-print. Non-print format specifies that no vertical format information is contained in the file.
MODE	The only supported mode is stream. Stream mode specifies that data is transmitted as a stream of bytes.
STRUCT	The only supported structure is file. File structure means that there is no internal structure and the file is considered to be continuous sequence of data bytes.
TYPE	ASCII and Binary data types are supported. When type is set to ASCII (default), data is transferred in standard 8-bit ASCII. When type is set to BINARY. data is sent as it appears on disc. FTP/1000, in Binary mode, will send internal record header and trailers of variable record length files as part of the data.

This feature set is identical with the UNIX implementation , therefore FTP/1000 is compatible with FTP on UNIX machines.

C. File System Differences

File systems under different operating systems have different characteristics. When using FTP, a user should have a knowledge of both the local and remote file systems in order to avoid confusion. For example, RTE is not case sensitive and file names are shifted to upper case, whereas in UNIX, test and TEST represent two different file names. A file name on RTE may have up to 16 alphanumeric characters. In addition, a file can have a file extension, marked by a dot followed by an extension of up to four characters. A file name in UNIX may have up to 256 alphanumeric characters and could contain several ".''s. Wild card characters may be different among different file systems. On RTE * is a valid character for a file name, whereas in UNIX, it is a wild card character. On UNIX @ is a valid character for a file name, whereas in RTE, it is a wild card character. Therefore, the user should always be aware of the target file system for the FTP command. For example, if a user has an FTP session to a UNIX system and wants to find all the files which start with "test", he/she would need to say

```
dir test*.*
```

The same request to an RTE system would be

```
dir test@.@
```

Some file systems, like the one on UNIX, are byte stream oriented and there are no file types. whereas, in RTE, file system is record oriented and there are different file types.

D. Binary vs ASCII Type

1. ASCII file transfer type is the default for FTP. ASCII file transfer should be used for transferring files containing ASCII data (i.e., file types 3 and 4). File types 1, 2, 5, and 6 usually contain binary data and will cause unpredictable results if you use ASCII file transfer. FTP/1000, in ASCII Type, assumes that the file being transferred is an RTE type 4 file. It will open/create the file as an RTE type 4, unless the file name contains the RTE type information (i.e. file1:::3). The default file descriptor for an ASCII file is:

File Type = 4
Maximum Record Length = 256 words
File Size = 24 blocks

FTP ASCII file transfers will use the above default file descriptor, unless the user specifies otherwise in the file specification.

2. Binary file transfer is used to transfer data as it appears on disc with no interpretation. On the HP-1000 binary transfers include some file system information such as record header and record trailer.

When using binary file transfer between HP-1000's, you must specify the file type of the target file to be the same as the file type of the source file. If not, the default target file will be set to type 1, and the result will be unpredictable if source and target file types do not match.

The default file descriptor for a binary file is:

File Type = 1
Record Length = 128 words
File Size = 24 blocks

FTP binary file transfers will use the above default file descriptor, unless the user specifies otherwise in the file specification.

Note that for file transfers other than file types 1 and 4, the file type specification is required to ensure that the target file has the same file type as the source file.

E. Miscellaneous Considerations

MPUT and MGET will let the user transfer multiple files with one command. Currently the HP-1000 does not handle an MGET request that requires the creation of subdirectories. To overcome this problem, the user should create the subdirectories before the MGET request.

When transferring files between HP-1000's, the best performance may be obtained by setting the transfer type to binary and specifying the file type in the command.

IV. TELNET

The TELNET protocol is a standard ARPA service that provides a virtual terminal connection to a remote node on the network. TELNET enables you to logon to remote nodes on the network as if you were on a terminal directly attached to the remote system. On the HP-1000, TELNET functionality is provided by four programs and a pseudo driver:

TELNET.RUN	TELNET User program
TNMON.RUN	TELNET call monitor
TNSRV.RUN	TELNET server program
HPMDM.RUN	RTE line monitor program
%IDZ00	TELNET pseudo terminal driver

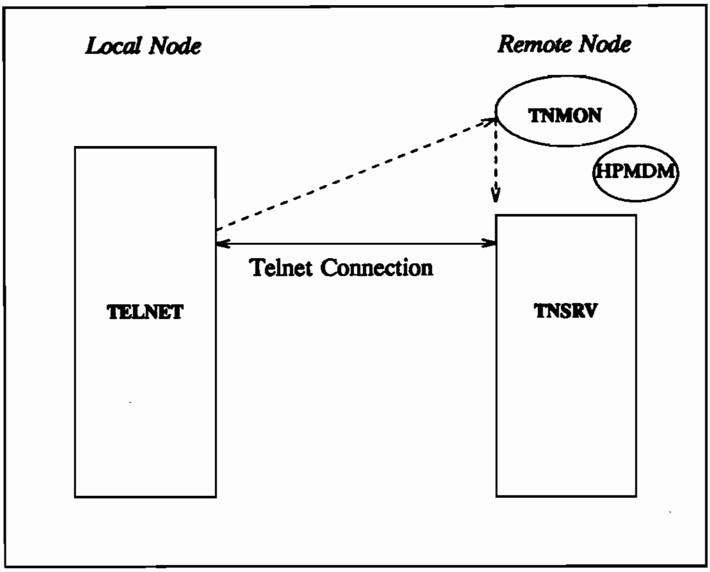


Figure 4: TELNET Processes

Figure 4 illustrates the interaction between these programs. Normally, a user starts a TELNET session by running the TELNET program:

TELNET *remote-host-name* or *remote-IP-address*

The TELNET program like FTP, sends a request to the remote node to establish a connection to a well-known TCP port number (23). On the remote node, the call monitor program (TNMON) is listening to TCP port 23. Upon reception of the connection request, TNMON completes the connection with the user program. Then it allocates the necessary resources and starts a copy of the server program (TNSRV), and hands off the connection to the server program. It then goes back to listen on port 23. The TNSRV then establishes its connection with the pseudo interface driver.

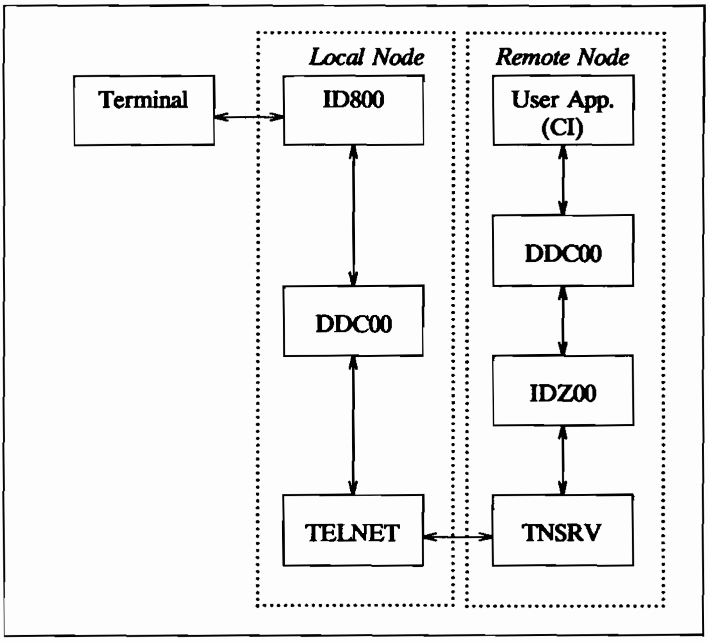


Figure 5: TELNET Data Flow

Figure 5 shows the different device and interface drivers involved in a TELNET session. The TELNET user program is always waiting to receive data either from the network or from the terminal. On the remote side, TNSRV is always waiting to receive data either from the network or from the pseudo interface driver. When a user hits a key on the terminal, a character is read from the terminal by the user program. Then, the user program sends the character to the server program on the remote node. The Server, gives the data to the pseudo interface driver, as if it were coming from a terminal device. On the remote node, a user application (i.e. CI) will receive the data through DDC00/01 driver. When a user application program wants to write data to the terminal, the same path will be traversed in the reverse direction.
HPMDM on the remote node is responsible for sending a termination notice to TNSRV when the session is terminated.

A. GEN/Setup Consideration

In order to accept telnet connections you need to gen %IDZ00 as an interface driver and associate some pseudo terminal devices to it via %DDC00/01. You should also customize HPMDM to work on these terminal LU's.
TELNET is supported only on terminals directly connected or connected by modem to the on-board MUX for A400 computers or to the 12040D MUX for other A-Series computers.
There are a couple of situations to keep in mind when using TELNET:

- In certain cases, it may take longer to send terminal data from the physical terminal over the network to the remote node than the time allowed by an application program. If the program fails to receive the needed data, it will result in an error. User written applications that are expected to run over TELNET should be written with this in mind.

- TELNET does not support 12040D Mux firmware with a revision earlier than 5.02. For connections to any computer, always set the HP-1000 host terminal RECVPACE configuration to XON/XOFF.

For further information on TELNET configuration consult the NS-ARPA/1000 Generation and Initialization Manual. For further information on TELNET commands consult NS-ARPA/1000 Users/Programmers Manual.

B. MiscellaneousConsideration

For best performance set the TELNET session into *LINE MODE*. This will cause the TELNET user program to do local echoing and transfer data to the server only when it senses a <cr> in the input stream. Some applications require TELNET to be in *CHARACTER MODE* (i.e. vi on UNIX). For more information consult NS-ARPA/1000 Users/Programmers Manual.

V. NetIPC to BSD Communication

NetIPC is HP's proprietary networking interface for interprocess communication. NetIPC Sockets available on all HP systems, and Berkeley Sockets, available on Berkeley's 4.3 UNIX (aka 4.3 BSD) operating system, have identical roles. Both provide a programmatic interface to the underlying network communication facilities. Sockets are an abstraction used by both Berkeley Sockets and NetIPC for network communication endpoints. This section assists the reader in creating inter-vendor applications, using TCP/IP and the two networking interfaces, NetIPC and Berkeley Sockets. While this is not officially supported by HP, it does work. Before creating an application, the developer should refer to the appropriate NetIPC or ARPA services manual for specifics on the use of the interfaces.

As mentioned before, NetIPC and BSD sockets are interfaces to underlying network protocols. Protocols determine interoperability, whereas interfaces determine portability. Of course an interface could make it difficult (or impossible) to have processes interoperate by not providing access to enough of the underlying protocols. For example, if *ipcname()* and *ipclookup()* were only intrinsics provided by NetIPC for address resolution, rather than providing *ipcdest()*, the NetIPC interface would not be sufficient to allow communication with BSD applications. However with the current release of NS-ARPA/1000 there are no significant roadblocks in allowing a process on a HP1000 to communicate with processes using BSD Sockets on other machines.

If the emphasis is on interoperability rather than portability, a one to one mapping of intrinsics from one interface to another is not necessary. NetIPC intrinsics such as *addopt()*, *initopt()*, and *ipcgive()* on RTE are strictly local issues and do not affect interoperability.

The problem of porting an application from UNIX to the HP-1000 is also complicated by differences in other, non-networking, system calls. For example, the "fork" call in UNIX to spawn a child process would have to be replaced by another call, such as RTE "EXEC" call, which may in turn cause some restructuring of the original code because of semantic differences between the calls. In general, porting applications will introduce problems other than simply mapping the syntax of different networking interfaces. Semantic differences between operating system calls may impact the structure of the application code.

We are concerned here only about interoperability, not portability. The primary issue for communication is compatibility of protocols, not the compatibility of interfaces. A byte stream passing through a LAN is the same whether the interface was NetIPC or BSD Sockets. This section focuses on illustrating the relationship between NetIPC and BSD Socket calls, and on providing a frame work for understanding the relative importance of protocols and interfaces.

A. CLIENT-SERVER MODEL

If we understand NetIPC and BSD Sockets as interfaces to the transport layer (TCP) rather than as end-to-end protocols in themselves, the conceptual problem of creating programs, one using NetIPC and one using BSD Sockets, is reduced. A model that is commonly used to describe network transport level communication is the client-server model. One process, the client, is requesting some operation from another process, the server. The client and server are also sometimes referred as the "active" and "passive" side respectively, because the client "actively" request a communication connection, whereas the server "passively" waits for those connection requests.

The client-server model is illustrated in figure 6. The column on the left identifies the operations being performed by the client or the server. The next two column of circles indicate the data structures held by the client or server, and the arrows between the circles indicate the direction and flow of information. As illustrated in the model, there are five distinct operation that are performed by the client and server, in the following order:

- Interface Setup: The data structures or "handles" required by the interface are created. These "handles" are analogous to the file system "file descriptors".

- Connection Establishment: The client process requests a connection with the server process and the server process "accepts" the connection request.
- Data Transfer: Data may be passed bidirectionally between processes over the newly established virtual circuit.
- Connection Release: Either the client or the server may shut down the virtual circuit, destroying the conduit for data between the two processes.
- Interface Release: The interface data structures local to each of the processes are destroyed. The interface release is analogous to the file system "close" operation.

CLIENT-SERVER COMMUNICATION MODEL

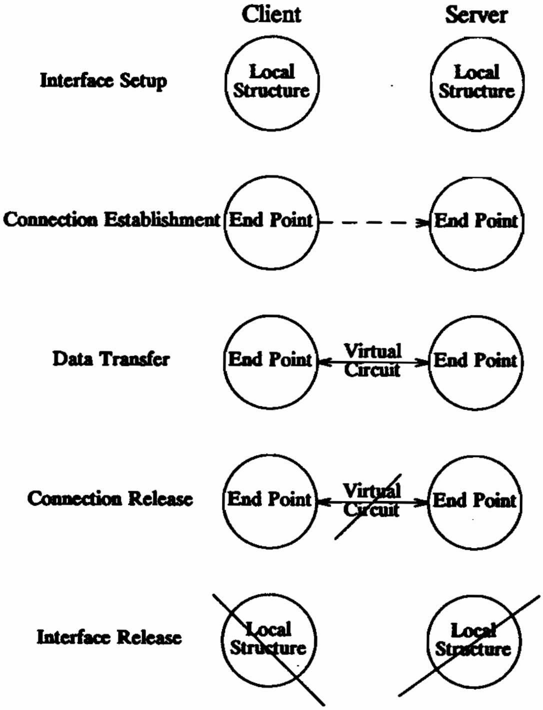


Figure 6: Client-Server Communication Model

A. NetIPC to NetIPC Communication

We can now map the NetIPC intrinsics onto the client and server operations as illustrated in figure 7. The NetIPC calls made by the client and server processes are laid out relative to the client-server model. The intrinsics on the left side of the model are made by the client process and the calls on the right side are made by the server process.

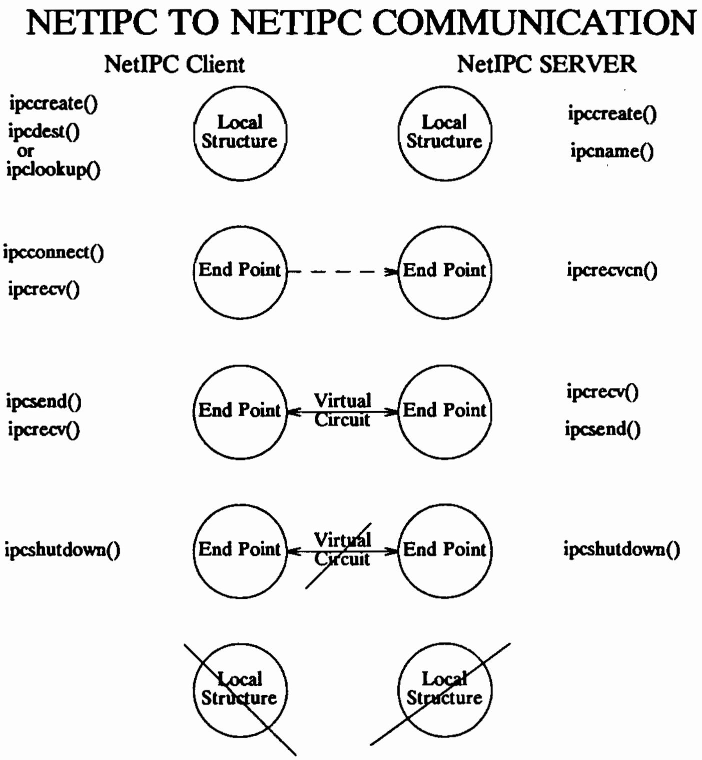


Figure 7: NetIPC to NetIPC Communication

The *ipccreate()* and the *ipcdest()* or *ipclookup()* calls create local "handles" to access the client process' communication socket. The *ipccreate()* called by the server creates a socket which the server will wait for a connection request on.

The "connection establishment" phase begins when the client process calls *ipcconnect()* to request a connection to the server process. The server accepts the connection request by calling *ipcrecv()*, and the client recognizes that the connection has been accepted when its first *ipcrecv()* call completes, completing the "connection establishment phase.

During the "data transfer" phase, the client and the server can pass data back and forth by calling *ipcsend()* and *ipcrecv()*. When either the client or the server process wants to terminate the connection, either process may call *ipcshutdown()* to initiate the "connection release" and "interface release" phases.

B. BSD Socket to BSD Socket Communication

BSD Sockets are mapped onto the client-server model in a way very similar to the NetIPC mapping. This mapping is illustrated in figure 8.

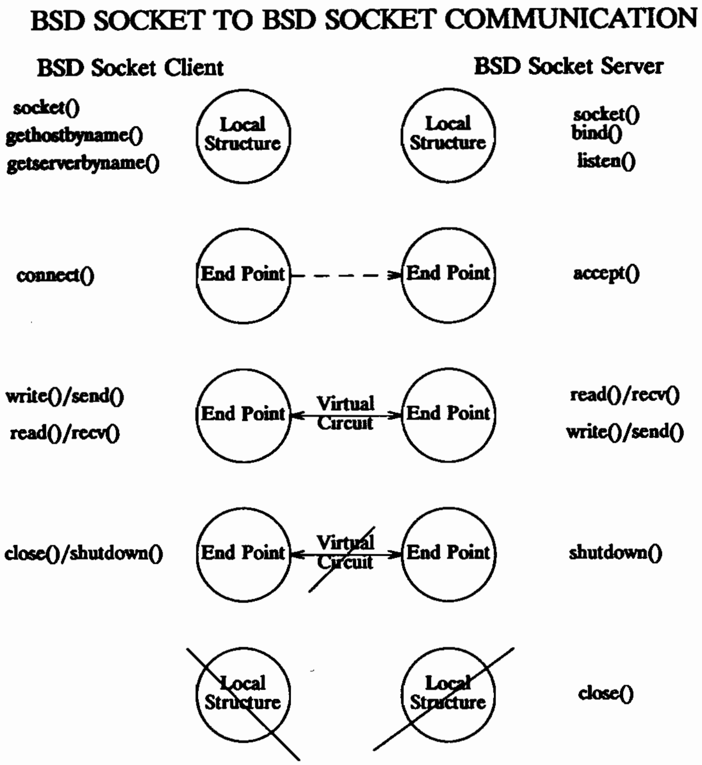


Figure 8: BSD Sockets to BSD Sockets Communication

A BSD client process will call *socket()* to create the local socket. The client will call the library routines *gethostbyname()* and *getserverbyname()* to create the address structure for the server host machine's IP address and the server's TCP port address, respectively. The BSD Socket server process will call *socket()* to create the local socket, and *bind()* to associate that socket with server's protocol address, which the client will request a connection to.

To begin the "connection establishment" phase, the client will request a connection to the server's socket by calling *connect()* and block until the server has accepted the request. The server will then call *listen()* to have the transport protocol accept connection requests on the server's socket. The *accept()* call transfers the connection accepted by the transport protocol to the server process, completing the "connection establishment" phase.

After the virtual circuit has been established between the client and the server, data can be transferred through the BSD Socket send and recv calls, or the process can use common UNIX file system calls, *read()* and *write()*. This makes up the "data transfer" phase.

Finally for the "connection release" and "interface release" phase, either the client or the server process can terminate the connection and cleanup the local data structures by calling BSD Socket *shutdown()* call or the UNIX file system *close()* call.

C. BSD Socket to NetIPC Communication

In order to develop applications which use different interfaces, it is important that each process, the client or the server, follows the semantics of the particular interface that is using, not the semantics of the interface the peer process is using. A simple illustration of a BSD Socket application talking to a NetIPC application is illustrated in figure 9. The client operations illustrated in figure 7 and the server operations illustrated in figure 8 have essentially been combined to form figure 9.

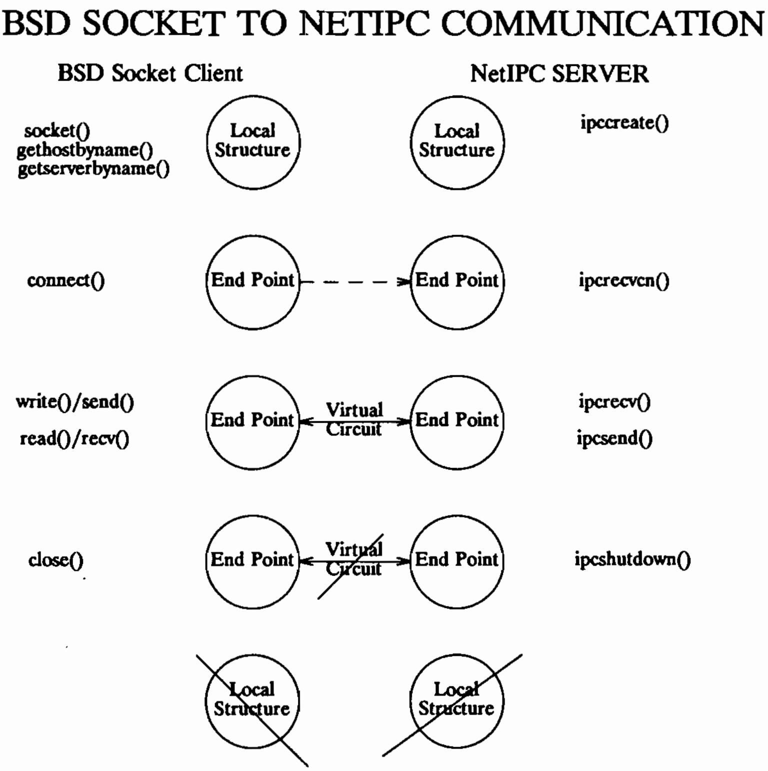


Figure 9: BSD Socket to NetIPC Communication

Note that NetIPC server must create its socket at a well known TCP port address rather than relying on *ipcname()* which uses HP's NS Socket Registry. The BSD Socket client would not find the NetIPC server if the server relied on *ipcname()*, since BSD Socket cannot access the NS Socket Registry. The NetIPC server must create its socket at a well known TCP port address during the *ipccreate()* call.

D. NetIPC to BSD Socket Communication

Similar to BSD Sockets Client to NetIPC Server illustrated in figure 9, a NetIPC client process connecting to a BSD Socket server process can also be done. This configuration is illustrated in figure 10. Note that the NetIPC client cannot use *ipclookup()* to find the server process by name. The client must use *ipcdest()* with a well known TCP port. Instead of *ipcname()* and *ipclookup()* which are not provided by BSD Sockets. Host addresses and server ports are configured in a file and extracted by library routines.

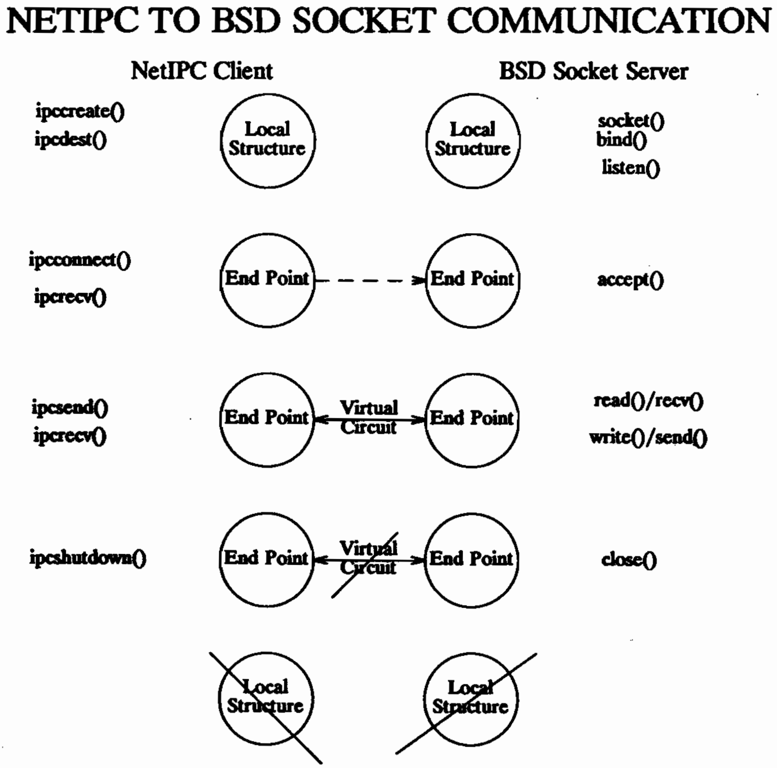


Figure 10: NetIpc to BSD Socket Communication

E. Guidelines

In summary, developing a NetIPC application to communicate with a BSD Socket application is not much more difficult than developing applications which use the same interface. In each case, the client or server process must make sure to follow the semantics of the interface that it is using.

Here are some broad guide lines in developing NetIPC to BSD Sockets:

- Although BSD Sockets and NetIPC have a broad set of intrinsics and options, the application developer should focus on those "core" intrinsics that are actually required for communication. Most other intrinsics are local in effect and will not impact interoperability.
- Core NetIPC intrinsics required for intervendor communication are:
  - *ipccreate()*
  - *ipcdest()*
  - *ipcconnect()*
  - *ipcrecvcn()*
  - *ipcrecv()*
  - *ipcsend()*
  - *ipcshutdown()*
- Core BSD Sockets intrinsics:
  - *socket()*
  - *connect()*
  - *bind()*
  - *listen()*
  - *accept()*
  - *send()*
  - *recv()*
  - *close()*

  The relationships between these "core" calls are illustrated in figure 7 through 10. Other calls, such as *addopt()*, *initopt()*, *gethostbyname()*, and *getserverbyname()* are strictly local issues and they are documented in the NetIPC and BSD programmers manuals. For more information refer to NS-ARPA/1000 Users/Programmers Manual and ARPA Services/300 User's Guide.

F. Technical Issues

The primary Technical issues involved in NetIPC to Sockets communication include:

- Compatibility of Protocols
  In general, if ARPA services work between the HP-1000 and the non-HP machine, the underlying transport protocols should be compatible, e.g. compatible TCP/IP, IEEE-802/ETHERNET.
- Addressing
  Only well known addresses can be used (i.e. ipcdest()),since BSD Sockets cannot access the NS Socket Registry, which is accessed through the ipcname() and ipclookup() intrinsics. Nodename to IP address resolution can be done through Probe protocol between HP machines, or, the mapping can be configured by NRINIT. This is roughly equivalent of the Unix /etc/hosts file.
- Functionality

  No "out of band data" interface provided through NetIPC- A work around is to create two VC's one for normal data, one for expedited data. ARPA services use this alternative for out of band data.

Checksumming-BSD UNIX implementations of TCP always have TCP checksumming on, whereas HP-1000 provides optional checksumming. The programmer must ensure that the use of checksumming is consistent for both client and server processes (NetIPC applications can enable/disable checksumming through flags in *ipcconnect()* and *ipcrecvcn()*). The initiating process (i.e. client) should always turn checksumming on. The server should not need to turn checksumming on because, if the initiating system has the checksumming on, the TCP protocol, on the HP-1000, will turn checksumming on automatically.

Access to caller's node and TCP port- At the time of the *accept()* call in BSD Sockets, The receiver can determine this information about the socket requesting the connection. This information currently is not available through NetIPC to general users.

No UDP- BSD Networking supports TCP and UDP protocols. Currently, UDP is not available in the NS-ARPA/1000 product.

- Configuration
  - Timeout Value
    Some timeout values may have to be adjusted through *ipccontrol()* due to different implementations of TCP/IP windowing policies. NetIPC Sockets have a default timeout value associated with them. You can turn it off so that NetIPC intrinsics will be blocked infinitely if necessary. You can set this timeout to a shorter or longer value if needed.
    BSD Sockets do not have a timeout value associated with them. They can be set to "Blocked" (infinite timeout) or "Non-Blocked" (zero timeout). You have to adjust your expectation on the result of the call based on the above differences.
  - Buffer Size
    NetIPC Sockets have a default socket buffer size of 100 bytes. BSD Sockets have a default socket buffer size of 4096 bytes. To optimize performance, use a socket buffer size at least twice as big as the size of your largest message. A simple thing to do is to enlarge the socket buffer from 100 bytes to 4096 bytes.

VI. Conclusion

NS-ARPA/1000 is capable of communicating with other vendors in an open networking environment. Currently, HP supports TELNET and FTP to the SUN and PC machines. Both TELNET and FTP implementations on the HP-1000 use NetIPC Sockets to communicate with BSD Sockets on SUN and PC machines. This shows that users should be able to develop multivendor network applications on the HP-1000 machines.

I would like to acknowledge Ron Morita whose paper on "NetIPC to BSD Socket communication" has been the basis for some of the material discussed in this paper.

Memory-Based NS-ARPA Systems for the HP 1000

Brian Barton / Erin Solbrig / Carolyn Krieg
Hewlett-Packard Co.
11000 Wolfe Road
Cupertino, CA 95014

Introduction

A memory-based system is a system in which the operating system and all programs are memory resident. A disk drive is not required for its operation. There is an important difference, however, between a memory-based system and a diskless system. In a diskless system the operating system requires a remote disk for its operation. Memory-based systems are often booted from a remote disk file, but once the boot process is complete, the system can run without access to a disk.

Memory-based systems are most often used in automated factories. There are two common reasons for using them. The system may be needed in an environment which is unsuitable for a disk drive. Vibration, dirt, or temperature extremes may be too harsh for a disk but not for a processor. A second reason for choosing a memory-based system is cost, a major factor in all design decisions. If the applications needed can fit in memory and the system can be booted over a network, the disk and its interface are unnecessary, thereby reducing the system cost. At least one system with a disk is necessary for a boot server and for application development. In a factory with many computer controlled cells the cost savings can be significant.

With the 5.2 release of RTE-A and NS-ARPA/1000, Hewlett-Packard has greatly strengthened its offering for memory-based HP 1000 systems. NS-ARPA/1000 and ARPA/1000 have been enhanced to run in memory-based systems. Also added to RTE-A for 5.2 is a RAM disk driver which allows portions of main memory to be reserved and then accessed as if it were a disk. This will allow programs which require local file access to run in a memory-based system. The BUILD program can now create a merged system file which includes RAM disks complete with files.

In addition, RTE-A now includes a download server and remote VCP that can operate over an 802.3 LAN. These functions have been available in the past with the DS/1000-IV and NS-ARPA/1000 products but only over HDLC DS links. The LANVCP software, which provides this download capability, does not require NS-ARPA/1000 for its operation.

This paper describes the new features that have been added for memory-based systems and their usage. It is divided up into the following three sections:

- *planning* an NS-ARPA/1000 memory-based system
- *creating* a memory-based system using BUILD
- *downloading* a memory-based system over LAN

Planning a Memory-Based NS-ARPA/1000 System

For a system designer planning a memory-based system, memory utilization will often be the foremost consideration. One will want to minimize the amount of memory required by RTE-A and NS-ARPA programs to reduce memory cost and to maximize memory available for applications. For NS-ARPA, this is accomplished by leaving out unnecessary programs and by properly sizing a few programs whose sizes are configuration dependent. Other considerations for the memory-based system designer include how to load program files residing on remote disks and the operation of NS-ARPA services in a non-multiuser environment.

NS-ARPA/1000 can be tailored to meet the needs of various systems by including only the programs necessary for the network links and services being used. In its full form, NS-ARPA/1000 includes three types of services and three different transports which can run over four network links.

NS-ARPA/1000 Services

The services available are the ARPA services, FTP and TELNET, the HP proprietary NS services, NFT and RPM, and the various DS Backward Compatible services, both RTE-RTE and RTE-MPE. The transports needed depend on the services and links used. TCP/IP is used for the ARPA and NS services. The DS RTE-RTE services use an older proprietary transport, but will encapsulate this in IP packets when used on a LAN link. The DS RTE-MPE services use another proprietary transport which is compatible with DS/3000.

A system designer must match the NS-ARPA/1000 programs included in the system to the networking requirements of the applications that will be used. For example, consider the four services NS-ARPA/1000 provides for accessing or transferring remote files. The following table summarizes the features of the four services. It is unlikely that a memory-based system would use all four of these services. The services used depend largely on the type of remote systems that will be accessed. If all remote file access will be to another HP 1000, then DS File Transparency may be the only file service needed. For file transfer with non-HP systems, FTP is required. NFT is an HP-proprietary file transfer service that works among the various HP systems. RFA is an older DS file access service that can access MPE files on DS/3000 systems and FMGR files on HP 1000 systems.

Summary of NS-ARPA/1000 File Access and Transfer Services

	Service			
	FTP	NFT	DS File Transparency	RFA
Type of Service	ARPA	NS	DS	DS
Systems	Multi-vendor	HP	HP1K	HP1K/3K
Access / Transfer	Transfer	Transfer	Access	Access
Subroutine Call	No	Yes	Yes	Yes

In addition to excluding programs for services that are not used, the system designer may also eliminate programs which are part of a needed service but only provide a feature that is not needed. If a memory-based system uses DS File Transparency to access remote files, the programs DSRTR and D.RTR are required. But if there will be no remote file access into the memory-based system, then the TRFAS program is not needed.

The Internal Resources section of the *NS-ARPA/1000 Generation and Initialization Manual*, part no. 91790-90030, lists the programs required for each type of network link and service. It also gives a brief description of each of the programs. To obtain a more complete understanding of the operation of NS-ARPA/1000, users are encouraged to read the Principles of Operation section of the *Maintenance and Principles of Operation Manual*, part no. 91790-90031. Provided below is a list of all the NS-ARPA/1000 programs with their sizes in pages.

NS-ARPA/1000 Program Sizes

Initialization	
NSINIT	30
NSPARS	33
NSPR1	52
NSPR2	49
NSPR3	46
MMINIT	7
NRINIT	43

Transports	
DSAM	151
INPRO	81
OUTPRO	79
UPLIN	22
TIMER	13
READR	2
QUEUE	2
GRPM	13
#SEND	9
MATIC	9
IFPM	18

Utilities	
NSINF	70
DSMOD	12
NRLIST	29
QCLM	11
BREVL	19
EVMON	28
LOGCHG	16
NSTRC	65
FMTRC	86
BRTRC	14
PING	40

NS-ARPA Programs	
DSCOPY	67
CONSM	53
NFTMN	41
PRODC	67
PRDC1	27
RPMMN	61
TNSRV	39
TELNET	68
TNMON	28
FTPSV	81
FTP	105
FTPMN	43

DS RTE-RTE Programs	
DSRTR	21
TRFAS	16
APLDR	11
DLIST	6
DSVCP	7
EXECM	9
EXECW	6
IOMAP	7
LUMAP	12
LUQUE	3
OPERM	6
PROGL	9
PTOPM	5
RFAM	12
VCPMN	4
REMAT	16
SYSAT	6

DS RTE-MPE Programs	
QUEX	8
QUEZ	3
RPCNV	9
RQCNV	9
CNSLM	4
DSTES	8
DSLIN	20
LOG3K	4
TRC3K	10
MVCP3	12
RMOTE	22

Once NS-ARPA has been initialized, the memory occupied by the initialization programs can be freed for other uses if there is never a need to shutdown NS-ARPA. The programs NSINIT, MMINIT, NSPARS, NSPR1, NSPR2, and NSPR3 use 217 pages of memory or 1569 blocks of RAM disk space and are only used for initialization and shut-down. If these programs are RP'd by BUILD, they can be OF'd after initialization. Other programs could then be assigned to the 11 reserved memory partitions. If they are placed on a RAM disk by BUILD, they can be purged after initialization, freeing up space on the RAM disk. If the RAM disk LU is not the first one initialized by BUILD, it is possible to deallocate the LU, returning the memory to the dynamic pool.

Sample NS-ARPA/1000 System Configuration

The following table shows the programs used and their sizes for a sample NS-ARPA configuration which includes the TELNET and DS File Transparency services. Besides the required transport programs, this system includes the error reporting programs EVMON and QCLM. The programs TRFAS and DSRTR are used by DS File Transparency to allow remote file access to and from this node. Since this is a DS service, the DS transport programs GRPM, MATIC, and IFPM are necessary. This system also allows TELNET connections both to and from it. TELNET and TNSRV have been placed on RAM disk so that they can be cloned as needed. The initialization programs have been placed on the RAM disk and will be purged once NS-ARPA is up. This configuration allows application programs in this system to use NetIPC routines for remote inter-process communication. The DS services, DEXEC, RFA, and PTOP (master only), may also be used by applications to access nodes with the appropriate monitors.

BUILT-IN PROGRAMS

		Program Size
Required Programs	**Program**	**(pages)**
Shared memory area	DSAM	151
Inbound protocol processor	INPRO	81
Outbound protocol processor	OUTPRO	79
Internal timing signaler	TIMER	13
Watchdog for clean-up and timeout processing	UPLIN	22
LAN link inbound message reader	READR	2
		348
Programs Required for DS Services		
DS transport monitor	GRPM	13
Interface monitor for DS over IP/LAN	IFPM	8
Message Accounting timeout processor	MATIC	9
HDLC/Bisync/X.25 inbound message reader	QUEUE	2
		32
Optional Programs		
Event logger	EVMON	28
DS transport and link error handler	QCLM	11
DS transparent file access master	DSRTR	21
DS transparent file access slave monitor	TRFAS	16
TELNET monitor	TNMON	28
		104

ON RAM DISK

Required Initialization Programs	Program	Program Size (pages)	File Size (blks)
Initialization main	NSINIT	30	232
Initialization parsing subordinate	NSPARS	33	256
Initialization subordinate 1	NSPR1	52	360
Initialization subordinate 2	NSPR2	49	344
Initialization subordinate 3	NSPR3	46	320
Memory manager initializer	MMINIT	7	57
		217	1569
Optional Programs on RAM disk			
TELNET user program	TELNET	68	512
TELNET server program	TNSRV	39	296
		107	808

The following table shows the amount of memory (in pages) used by NS-ARPA programs while NSINIT is executing and when NSINIT has completed and the initialization programs have been purged from the RAM disk. After initialization, each TELNET connection into this node will use one copy of TNSRV (39 pages), and each copy of the TELNET user program being run will need 68 pages of memory.

	Reserved Partitions	Dynamic	RAM Disk	Total
During Initialization	484	217	298	999
After Initialization	484	0	101	585

One can see from the above numbers that this configuration could not run in a system with 2 Mbytes of memory. It is possible to configure NS-ARPA/1000 to run in a 2-Mbyte memory-based system, however, such a system would leave very little memory available for optional services or user-written applications.

Reducing NS-ARPA Program Memory Usage

The size of DSAM, the NS-ARPA shared memory area, can be adjusted to match different configurations and usage levels. Its size is set by the EM command when linking MMINIT. The size of the memory partition must be one page larger than that specified in the EM command to allow room for the EMA page table. Thus, the default SHEMA size of 150 pages requires a 151-page partition. Once NS-ARPA is running, one can determine if the size of DSAM is insufficient or excessive by examining the memory manager statistics with the NSINF B command. Errors in the NS-ARPA log file may also indicate that DSAM is too small. It is difficult to calculate the proper size for DSAM, but the Internal Resources chapter of the *NS-ARPA/1000 Generation and Initialization Manual* gives guidelines for calculating the buffer space used by NS-ARPA services and NetIPC application programs. The fixed table space required in DSAM is reported by NSINIT in its output file.

The EMA size of the NSINIT subordinate programs, NSPR1, NSPR2, and NSPR3, is also configuration dependent. This EMA is used to set up tables before moving them to DSAM. The load files specify three pages of EMA for each program. In most systems one page of EMA is sufficient. NSINIT prints, in its output file, the amount of EMA used by each program. With this information NSPR1, NSPR2, and NSPR3 can be relinked to reduce the EMA size to the amount needed.

Loading Programs

With NS-ARPA/1000 there are two mutually exclusive ways to load programs from a remote disk into a memory-based system. The first option is simply to load programs from the RAM disk. The program file can be copied onto a local RAM disk using any file transfer service. The RAM disk must have enough free space to accommodate the file. When the BUILD program initializes a RAM disk LU it allocates space only for the files it places on the RAM disk, leaving no free space. To obtain free RAM disk space, a RAM disk partition can be created in dynamic memory after boot-up, or files may be purged from a built-in RAM disk. From the RAM disk a program can be RP'd and loaded into memory in the same way as in a disk-based system.

The second option for loading remote programs is to use APLDR. This monitor program is part of the DS/1000-IV Backward Compatible Services. It can be invoked by the REMAT LO command or the FLOAD subroutine to load a program from a FMGR file into a memory-based system. The node issuing the request, the node where the file resides, and the memory-based node may be three different nodes. APLDR uses RFA to read the remote program file and loads it directly into memory. A RAM disk is not required. There are four restrictions, listed below, which currently limit APLDR's usefulness.

1. **The operating system cannot contain the modules LOAD or MEMRY.** Only APLDR can be used to load programs and manage memory. The CI RP command cannot be used and the RU and XQ commands can be used only for programs already loaded into memory. This also applies to the subroutines FmpRpProgram and FmpRunProgram. Because of this, all programs scheduled by NS-ARPA software must be built-in. This includes the initialization programs and server programs.

2. **APLDR does not use dynamic memory.** It can only use the fixed memory partitions created by BUILD. Since dynamic memory is not available, RAM disk volumes cannot be dynamically allocated or deallocated. All RAM disk volumes must be set up by BUILD.

3. **APLDR can only access program files on FMGR volumes.** All programs that are to be loaded by APLDR must exist as type 6 files on FMGR volumes. Since the BUILD program creates only CI volumes, APLDR cannot load programs placed on RAM disk by BUILD.

4. **APLDR cannot load CDS programs.** The version of APLDR provided by NS-ARPA/1000 does not have the capability to load CDS programs.

To use APLDR, it and EXECW must be present in the memory-based system where the program is to be loaded. RFAM, the RFA monitor, must be running in the node where the program file exists. The load can be initiated from any node by the REMAT LO command or the FLOAD subroutine.

Considering the restrictions on the use of APLDR, loading programs from RAM disk will be preferable for most systems. However, it has the drawback of requiring one copy of each program in RAM disk memory, in addition to any copies executing in dynamic memory. A method for loading programs directly from remote CI files into dynamic memory is needed.

NS-ARPA File Usage

In a disk-based system, NS-ARPA/1000 uses a variety of disk files. These include input command files, log files, error message files, and scratch files. In a memory-based system, users should consider alternatives to disk files, since RAM disk space is limited. LU numbers can be substituted for input or output file names. After initialization, remote files can be accessed via DS File Transparency. When error message files are missing, programs will display only error numbers. With the numbers, the messages can be found in the *Error Messages and Recovery Manual*, part no. 91790-90045. The files normally used by NS-ARPA/1000 are described below along with alternatives to placing them on a RAM disk.

1. **NSINIT input file.** This file contains the responses NSINIT uses to initialize NS-ARPA/1000. NSINIT can read its responses from an LU such as a terminal, but this would require some typing. For convenience, this file should be placed on a RAM disk.
2. **NRINIT input file.** This file is read by NRINIT to initialize the nodal registry. With a probe proxy server on the LAN, it is not necessary to initialize the nodal registry. Since NRINIT is run after NSINIT, a remote file may be used here.
3. **Event log file.** NS-ARPA/1000 uses this file to log events such as errors. This file should not be placed on a RAM disk volume since it grows as errors are logged and because RAM disk files are as volatile as the RAM. HP recommends that log messages be sent to an LU such as the system console on memory-based systems. NSINIT will accept an LU number for a log file name. With DS file transparency, a remote file can be used, but then EVMON cannot record errors when the remote file access fails. This can lead to further complications. Event logging can be disabled by entering /E when NSINIT asks for event classes to log.
4. /SYSTEM/NSINIT.MSG. NSINIT reads its error messages from this file. Without it, NSINIT will only display numeric error and warning codes.
5. /SYSTEM/NSERRS.MSG. This file contains error messages used by the NFT, TELNET, and FTP user programs. The messages are read into DSAM at initialization and the file is not needed afterwards. If not present, only error numbers are displayed.
6. /SYSTEM/NODENAMES. This file is used by DSRTR to map nodenames to DS node numbers. Without it, only node numbers can be used to specify remote files with DS file transparency.
7. **Help files.** The files DSCOPY.HLP, FTP.HLP, and TELNET.HLP, are used by those programs when users enter the ? command. Without these files, DSCOPY, FTP, and TELNET will still operate, but will not be able to provide help on their commands. Other help files are placed in the /HELP directory and are listed in response to the CI ? command to provide runstring information.
8. /SCRATCH.DIR. FTP uses this directory for temporary files when processing directory list commands and commands that use file masks. Without a /SCRATCH directory, these commands cannot be used.

Memory-Based vs. Disk-Based Systems

When moving an application from a disk-based system to a memory-based system there are a few differences a programmer must be aware of. These differences also affect the operation of some NS-ARPA services. Programs which are built-in remain in memory after completion so that they can be run again. This causes static variables in the data segment to retain their values from one execution to the next. Another difference is that RTE-A can only clone programs from a type 6 program file. It cannot clone from a program in memory.

Since most memory-based systems will not have multi-user capabilities, those accustomed to using a multi-user system need to understand the differences between multi-user and single-user systems. The VC+ product contains the programs necessary for a multi-user environment in addition to the modules required to run CDS programs. While the multi-user programs may be undesirable for a memory-based system, the CDS modules are required to run NS-ARPA/1000.

The PROMT program determines whether a system is multi-user or not. In a multi-user system, PROMT is normally enabled as the primary program for all terminal LUs. The first time it runs, PROMT will initialize the multi-user system. It RP's LOGON and CI (calling it CM) if they are not already RP'd. Thus a multi-user system requires the programs PROMT, LOGON, and CM, and a /USERS directory with the files LOGONPROMPT, MASTERACCOUNT, MASTERGROUP, and any user account files.

Using NS-ARPA Services in Memory-Based Systems

The services FTP, NFT, and TELNET all operate in a similar manner. The user initiates a request from the user program. It then establishes a connection with the appropriate monitor program at the remote node. The monitor then clones a copy of the server program and passes the connection to it. The user program then communicates with the server to fulfill the user's request while the monitor waits for new connections. NFT is more complex than FTP and TELNET because it can operate between three nodes. It is described in detail in the *Maintenance and Principles of Operation Manual*.

In any node there is always only one copy of the monitor program for each service. So it is always advisable to build in any monitors that will be used in a memory-based system.

New copies of the user and server programs are needed for each connection. If the type 6 or .RUN file is placed in the /PROGRAMS directory on a RAM disk then these programs can be cloned just as they are in a disk-based system. These programs can also be RP'd into your memory-based system by the BUILD program with some restrictions. Because the monitor programs will only use an RP'ed server program if its name has not been modified, only one server program for each service should be built in. For the user programs, multiple copies can be RP'd by BUILD as long as each one has a unique name. That name must then be used to schedule it.

Service	User Program	Monitor	Servers
TELNET	TELNET	TNMON	TNSRV
FTP	FTP	FTPMN	FTPSV
NFT	DSCOPY	NFTMN	CONSM, PRODC, PRDC1

The NS-ARPA services, NFT, FTP, and RPM, all use sessions to provide access at remote systems. When an NFT or FTP request arrives at a single-user system, logon names and passwords are not checked. Access is given to all requests. The NFT and FTP server programs must execute in the system session and, therefore, they give users full access into the non-multiuser system. RPM, however, does not allow access into a single-user system. Its design uses sessions extensively and it has not been modified to operate in a system where user sessions cannot be created. Thus, the RPM monitor program, RPMMN, cannot perform any function in a system without multi-user capabilities. Programs in a single-user system can call RPM subroutines to create sessions and control programs in other nodes, but the reverse is not possible. Using RPM terminology, a single-user system can be a parent node but not a child node.

TELNET and HPMDM in Single-User Systems

Besides displaying messages on the system console for each connection and disconnection, HPMDM does two things. When a user logs off, HPMDM is scheduled by LOGON to close the TELNET connection. Also, whenever a TELNET connection is closed, the TELNET pseudo-driver schedules HPMDM, which cleans up any leftover user sessions. For a system without user sessions, the only service HPMDM provides is the messages displayed on the system console. Since the source for HPMDM is provided with RTE-A it is possible to modify HPMDM to fit a particular application. Another option is to simply leave HPMDM out of the system.

The default primary program that is scheduled on an interrupt from a terminal or TELNET LU is PROMT. PROMT cannot be used in a non-multiuser system, so the primary program must be set to another program. The following steps demonstrate how to make a copy of CI for the primary program and use CM as the secondary program for a TELNET LU.

1. RP a copy of CI for each TELNET LU giving each one a unique name. Also, RP one copy of CI with the name CM. This can be done in the BUILD command file or in the welcome file if the program file CI.RUN is present on the RAM disk.

2. Enable the appropriate copy of CI as the primary program for each TELNET LU with a CN,lu,20B command. If HPMDM is being used, it must be run to enable the primary program.

3. Use a CN,lu,40B command to enable CM as the secondary program for each TELNET LU.

When a TELNET connection is established to the system, the TELNET server responds with its usual message, shown below.

```
***    Welcome to TELNET Server !!    ***
Please hit a <cr> to get the logon prompt.
```

The first key entered causes the pseudo-driver to schedule the primary program. Thus if CI is used, its prompt will be displayed. When the user is finished using the TELNET connection it must be explicitly closed. To do this, the user must enter the TELNET escape character (default is ^]) to get the TELNET> prompt, then enter the command CLOSE or QUIT.

Creating a Memory-Based NS-ARPA/1000 System

This section is designed to step the user through the generation and installation of a memory-based NS-ARPA system. Throughout this section, the server refers to the system with a disk. The client refers to the memory-based system which will be created in this section.

There are four steps to creating a memory-based node which are outlined in the following sections:

1. Generate the server system and initialize NS-ARPA/1000.
2. Generate the client system with VC+, NS-ARPA/1000 drivers, and a RAM disk.
3. Link all NS-ARPA/1000 programs and other programs with the client snap file. Create the CLIENT.NSIN, CLIENT.NRIN, and welcome files for the client.
4. Build the client system and boot the client. After the client is booted, create a dynamic RAM disk LU and initialize a /SCRATCH directory.

The generation of a LAN link and the following NS-ARPA/1000 services are included:

Network File Transfer (NFT)
DS/1000-IV Compatible Services
ARPA - Telnet and FTP

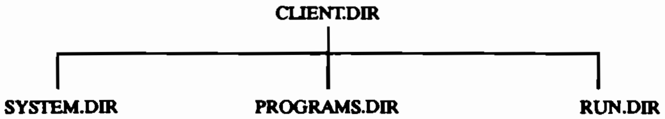
The following are not included:

X.25
Remote Database Access

Generating the Memory-Based Client System

Generate a 5.2 RTE-A system and with your network planned, install and initialize the NS-ARPA/1000 services needed for your system. See /NS1000/EXAMPLES/#ANSNS and the *NS-ARPA/1000 Generation and Initialization Manual*, part no. 91790-90030.

1. Create a /CLIENT directory with the following subdirectories: This directory can become quite large, since it will contain all of the client programs.



This will keep the disk-based and memory-based NS-ARPA and RTE-A programs separate.

2. Set your working directory to /CLIENT/SYSTEM and create an answer file for the client with the following modules and drivers:

   - Required modules and drivers for NS-ARPA/1000. See the *NS-ARPA/1000 System Generation and Initialization Manual* for more information.

     NOTE: If this system is to be downloaded and booted using LANVCP from the server system, the select code of the client LAN card must be set to 24 (octal) if the client system is configured for autoboot.

   - Required modules and drivers for your application. Because memory is limited, it is important to remove any device IFT and DVT entries that are not needed.

   - The RAM disk driver as follows:

     Relocate the interface driver in a driver partition; there is no device driver.

     Define an IFT entry and DVT entries for at least two RAM disk LUs. Because BUILD does not leave any free space on LUs it initializes, a second LU is required for creating new files. Note that size is not specified. RAM disk size is determined during BUILD (if the disk LU is initialized before boot time.) If the LU is initialized after boot time, size is determined by issuing control commands (CN lu 25B blocksize) at the CI> prompt after the memory-based system is up and running.

```
ift,idr37.rel

dvt,,,dx:8,dt:33b,lu:2
dvt,,,dx:8,dt:33b,lu:3
```

3. Run the generator on CLIENT.ANS as follows:

```
CI> rtagn client.ans - - -
```

Linking the RTE-A and NS-ARPA Programs for the Client

1. Link the required RTE-A programs with the client snap file, CLIENT.SNP, and place the programs in /CLIENT/PROGRAMS.

```
CI> wd /rte_a
CI> link #ddrtr /client/system/client.snp /client/programs/ddrtr.run
CI> link #dderr /client/system/client.snp /client/programs/dderr.run
CI> link #cia   /client/system/client.snp /client/programs/ci.run
CI> link #cix   /client/system/client.snp /client/programs/cix.run
```

   Other useful programs for the client that can be linked at this time are:

Program	Load File
SPORT	/RTE_A/SPORT.LOD
FREES	/RTE_A/FREES.LOD
WH	/RTE_A/WH.LOD
DL	/RTE_A/#DL
IO	/RTE_A/IO.LOD
LI	/RTE_A/LI.LOD

2. Set the following CI local variables to load the NS-ARPA/1000 programs for the client system. Note that these variables are not permanent and must be reset each time you log off.

```
CI > set ns_snap = /client/system/client.snp
CI > set ns_cdir = /client/programs/
CI > set ns_sdir = /client/programs/
CI > set ns_hdir = /client/system/
CI > set ns_helpdir = /client/system/
CI > set ns_idir = /client/system/
CI > set ns_mdir = /client/run/
```

3. Edit the install command file for NS-ARPA and uncomment the lines to link TRFAS and DSRTR with the NS-ARPA libraries.

```
CI > wd /ns1000
CI > edit cmd/install_ns1000.cmd
```

See the file /NS1000/CMD/INSTALL_NS1000.CMD for more information about its usage. To load the NS-ARPA/1000 software, set the working directory to /NS1000 and transfer to CMD/INSTALL_NS1000.CMD.

```
CI > tr cmd/install_ns1000
```

4. Set the working directory to /RTE_A and load READR and the Node Manager (NM, NM2, and NMGR) programs.

```
CI > wd /rte_a
CI > tr lan8023.cmd /client/system/client.snp /client/programs
```

5. Also, load the HPModem program (HPMDM) which is used for TELNET services. HPMDM displays connection and disconnection messages to the system console. It also terminates a user session when TELNET or a modem connection is closed. If the system is a single-user system or has no system console, then HPMDM is not required.

```
CI > link #hpmdm /client/system/client.snp /client/programs/hpmdm.run
```

6. Using the network worksheet for client, run NSINIT and build the output file (CLIENT.NSIN) and put this file in /CLIENT/SYSTEM.

```
CI > /ns1000/run/nsinit
```

To initialize the Nodal Registry, simply use the server's NRINIT input file.

```
CI > co /system/server.nrin /client/system/client.nrin
```

BUILDing the Client System

1. Create a welcome file for the client (/CLIENT/SYSTEM/WELCOME9.CMD). The welcome file is required when CI is the start-up program and networking services will be initialized during boot up. In this case, the runstring for NSINIT must be included in the welcome file. NRINT can also be run in the welcome file at boot time. If optional services such as TELNET, HPMDM, or IOMAPping are generated into the system, then they also can be configured in the welcome file.

   Run NSINIT and NRINIT on the respective client input files, but remember that the client NSINIT and NRINIT input files will be located in the client's /SYSTEM directory once the client is booted. (NRINIT can be run on the SERVER.NRIN file by using DS-transparency. In this example, CLIENT.NRIN is built onto the RAM disk.)

   ```
   /programs/nsinit /system/client.nsin
   /programs/nrinit /system/client.nrin,,r
   ```

   If you have generated TELNET and HPMDM into your system, ADd any TELNET LUs to HPMDM and configure HPMDM to have a copy of CI as the primary program for each TELNET LU. In this client system, there is only one TELNET LU.

   In /CLIENT/SYSTEM/WELCOME9.CMD, add the following commands to initialize HPMDM.

   ```
   hpmdm lu ad ok
   hpmdm lu pr=ci1
   ```

   Configure the TELNET LUs to have a separate copy of CI as the primary program. If there is more than one TELNET LU, there must be more than one copy of CI RP'd during BUILD.

   ```
   cn, lu 20b ci1
   cn lu 40b cm
   ```

   If your system has IOMAPping included, initialize it here:

   ```
   iomap lu -1
   echo $return1
   ```

2. Create an input command file for BUILD (CLIENT.CMD). A sample BUILD command file is given at the end of this section. This file is located in the /CLIENT global directory. Use the CLIENT.CMD input file to run BUILD with the +D option.

   ```
   CI> build +d client.cmd
   ```

3. After the system is successfully built, it can be booted from any supported VCP loader, for example, downloading using LANVCP.

4. After the client system is booted, the RAM disk LU not initialized by BUILD should be allocated, initialized, and a /SCRATCH directory created. To do this, the size of physical memory to be dedicated to the RAM disk must be determined. To determine the size of the "left over" memory not used by the system or the initialized RAM disks, run WH,PA. The number of pages for the RAM disk can be divided by pages/track to obtain the number of tracks available on the disk. For example, a system with 600 pages available will yield a disk LU with 150 tracks (at 4 pages/track or 32 blocks/track). See the *RTE-A Driver Reference Manual*, part no. 92077-90011, for more information.

   The RAM disk must be configured before it is used. This is done by issuing a CN command, either interactively or in the welcome file. The configuration command is:

   ```
   cn <lu> 25b <pages> [<pages/track>]
   ```

   where: lu — is the LU number of the RAM disk.

   pages — is the number of pages of RAM disk memory.

   pages/track — is the number of pages per track (1 through 8). The default is 4 pages/track.

   For example, to configure a RAM disk LU of 600 pages, the command is:

   ```
   cn <lu> 25b 600
   ```

   This would use the default of 4 pages/track, thus the disk would have 150 tracks.

   Now the disk is available for access. It can be mounted and directories can be created. RAM disk memory can be returned to the system by issuing the following control request:

   ```
   cn <lu> 26b
   ```

   Disk memory must be returned to the system if the RAM disk size is to be changed. However, by returning disk memory to the system, the RAM disk files are destroyed. Therefore, care must be used when determining the RAM disk size from the beginning, because the size cannot be changed without destroying the files on the LU.

Sample BUILD Command File

This is the client's sample input command file for BUILD. The comments surrounded by '*' are for reference only. It is assumed that this file is located in the global directory /CLIENT and that all other files are located in the SYSTEM/, RUN/, and PROGRAMS/ subdirectories.

The system file DWNLOAD can be put on LU 16 so that it can be easily tested for correctness by booting the system from disk first. However, before this can be done, all files except BOOTEX should be purged from LU 16 so that DWNLOAD is located immediately following BOOTEX. This sytem can be booted with the following runstring:

```
VCP> %bdc300<select code>
```

After the system is built and working, move the file into the /FILES802 directory and run IPL_BUILD to create the table needed for RMVCP. Also, a file size must be specified for DWNLOAD since the default file size is too small.

```
DWNLOAD::16:1:25400
system/CLIENT.SNP
system/CLIENT.SYS
yes,,Automatic partitioning
3800,,system size
rp,programs/ddrtr.run,d.rtr,,RP RTE modules
rp,programs/dderr.run,d.err
rp,run/dsrtr.run,, DS Transparency
rp,run/trfas.run
rp,programs/ci.run
rp,programs/ci.run,cm,,
********************************************************************
* RP a copy of CI for each TELNET LU
********************************************************************
rp,programs/ci.run,ci1
rp,programs/ci.run,start
st,,9,,,                Use welcome9.cmd
rp,run/timer.run,,      NS-ARPA required modules
rp,run/inpro.run
rp,run/outpro.run
rp,run/uplin.run
rp,run/mminit.run
rp,run/nspars.run,,     Required for NSINIT
rp,run/nspr1.run
rp,run/nspr2.run
rp,run/nspr3.run
rp,programs/readr.run,read1, IEEE 802.3/Ethernet
rp,programs/readr.run,read2
rp,run/matic.run,,      Message Accounting
rp,run/queue.run,,      RTR/DS1000-IV/Gateway Half
rp,run/qclm.run,,       RTR/DS1000-IV/Gateway Half
rp,run/grpm.run,,       RTR/DS1000-IV
rp,run/ifpm.run,,       DS1000-IV
rp,run/tnmon.run,,      Telnet monitor
rp,run/evmon.run,,      Event monitor
rp,programs/hpmdm.run,, Hp Modem, required for Telnet
rp,run/nftmn.run,,      Network File Transfer - NFT
rp,run/ftpmn.run,,      FTP Monitor
rp,run/#send.run,,      Dynamic Rerouting
rp,run/execm.run,,      Distributed EXEC/REMAT
rp,run/execw.run,,      Distributed EXEC/REMAT/Remote Prog. Download
```

```
rp,run/apldr.run,,      Remote Program Download
rp,run/dlist.run,,      REMAT
rp,run/ptopm.run,,      Prog-to-Prog Communication
rp,programs/remat.run,, REMAT
rp,run/iomap.run,,      I/O Mapping
rp,run/lumap.run,,      I/O Mapping
/e
2,mc
system/nsinit.msg
system/nserrs.msg
system/dscopy.hlp
system/welcome9.cmd
system/client.nsin
system/client.nrin
/e
*********************************************************************
* The above files will go in the /SYSTEM directory on the client.
*********************************************************************
system
run/nsinit.run
run/nrinit.run
programs/ftp.run
programs/telnet.run
programs/dscopy.run
programs/brtrc.run
programs/fmtrc.run
programs/nrlist.run
programs/nsinf.run
programs/dsmod.run
programs/prodc.run
programs/prdcl.run
programs/tnsrv.run
programs/ftpsv.run
programs/wh.run
programs/io.run
programs/li.run
programs/dl.run
programs/ci.run
programs/cix.run
programs/frees.run
programs/sport.run
********************************************************************
*If the Node Manager program is not needed, omit the following three
*lines and do not create the /FILES802 directory below.
********************************************************************
programs/nm.run
programs/nmgr.run
programs/nm2.run
/e
********************************************************************
* The above files will go in the /PROGRAMS directory
********************************************************************
programs
/files802/menu
/e
files802
/e
/e
```

Downloading a Memory-Based System with LANVCP

This section gives procedures for downloading a memory-based system over LAN to a remote system. Throughout this section, the server refers to the system in control, the disk-based node which is running the download monitor program, VCPMT. The client refers to the system to which the memory-based system is being downloaded.

The download operation can be initiated using one of the following three methods:

- **Interactive Download** - Using the RMVCP (Remote VCP) program from the server, you can remotely access the VCP of the client and send it a bootstring to initiate a download and boot over the LAN. The LAN card in the client system must be enabled for VCP; therefore, the client will not have a VCP console.
- **From the Client's VCP Console** - Press BREAK on the client's system console to get the VCP prompt. Enter the "%BDS..." bootstring giving the select code of the LAN card. In this case, the LAN card in the client is not enabled for VCP.
- **Autoboot** - The client can be configured for autoboot (by setting the CPU switches) over the LAN. Upon a power-up sequence, the client's VCP will request a download. In this case, it does not matter whether the client's LAN card is enabled for VCP; however, the select code of the client's LAN card must be set to 24.

After the client's VCP receives a boot command in one of the above three ways, it then sends a command to its LAN card. The client's LAN card firmware then sends a download request message to the download server using the Download Server Address that has been set on the client's LAN card. The download server program, VCPMT, receives the request from the client and determines which system file to download to that particular client. The server then begins the block-by-block transfer of the system file to the client's VCP.

Preparing your memory-based system for downloading over LAN consists of the following steps which are described in the subsections that follow:

1. Set the client Download Server Address and determine the client's LAN Station Address.
2. Link the LANVCP software programs.
3. Run IPL_BUILD to create your configuration file.
4. Configure the client hardware.
5. Invoke VCP of the client and initiate the download and boot.

The LANVCP software is compatible with the use of downloadable system files that have the P*fffff* file naming convention. This naming convention is currently used with the DSVCP product. Refer to the subsection "Selecting the System File to Download" later in this section for a detailed description of how the LANVCP software uses P*fffff* files.

Client Addressing Configuration

Before linking the LANVCP programs and building the configuration file that is used during the download process, you must first determine the Download Server Address to set on the client's LAN card. All LAN cards, when shipped from the factory, are set to use a broadcast address for downloads. It is recommended that this be reconfigured for individual addressing. For a detailed description of the differences between individual/multicast/broadcast addressing and how to implement them on your node, refer to the *LAN/1000 Node Manager's Manual*, part no. 12076-90002. Also, you must obtain the client's Station Address for use on the server system.

To configure the client's LAN card for individual addressing, do the following:

1. Run the Node Manager (NM) program to obtain the Station Address of the server's LAN card (this address has also been written on the LAN card itself). Write down the Station Address of the server's LAN card for use in the next step.

   ```
   CI> nm
   NM> rc,,,,<lu>
   ```

   where <lu> is the even-numbered LU of the server's LAN card.

2. Replace the server's LAN card with the LAN card that will be used in the client system. Reboot the server system ensuring that the U1 switch settings on the client's LAN card are set to the select code of the server's LAN card that was just removed.

   Run the NM program again to set the Download Server Address on the client's LAN card:

   ```
   CI> nm
   NM> sc,,9,<download server address>,p
   ```

   where <download server address> is the Station Address of the server system obtained in the previous step.

   This address is stored in non-volatile memory on the LAN card so that it does not have to be reset when the card is removed from the server system and installed into the client.

3. Run the NM RC command to get the Station Address of the client (using the same LU as above):

   ```
   NM> rc,,,,<lu>
   ```

   Write down the Station Address of the client for use when running IPL_BUILD.

4. Remove the client's LAN card and reinstall the server's LAN card into the server system. Turn the power back on and reboot the server system.

Linking the LANVCP Programs

Use the command file INSTALL_VCP.CMD to link the LANVCP programs as follows:

1. Set your working directory to /VCPLUS/LANVCP and transfer to the command file INSTALL_VCP.CMD to link the LANVCP programs:

   ```
   CI> wd /vcplus/lanvcp
   CI> tr install/install_vcp.cmd [snap] [number]
   ```

   where [snap] is the name of your snap file. This parameter is optional and defaults to the current system snap file.

   [number] the number of clients that you expect to connect to your server. This number is optional and defaults to 16. The maximum is 28.

   Note that the relocatables used by INSTALL_VCP.CMD are located in /VCPLUS/LANVCP/REL. The load files are located in /VCPLUS/LANVCP/LOD. During execution of INSTALL_VCP.CMD, the run files are placed in /VCPLUS/LANVCP/RUN and also copied to the /PROGRAMS directory.

2. Transfer to BOOT_VCP.CMD. This command file begins execution of VCPMT (VCP Monitor) and DISPATCH (Dispatcher). The user LU of the LAN card that VCPMT and DISPATCH will monitor must be specifed. This command can also be put in your system welcome file for execution at boot-up.

```
CI> install/boot_vcp.cmd <lu>
```

Running IPL_BUILD

The IPL_BUILD program is used to generate the configuration file that is used during the download process. VCPMT uses this file along with the client Station Address, received in the download request, to determine which system file and LAN interface to use for the download. The default configuration filename is /FILES802/IPL_TABLE.TXT.

IPL_BUILD does not change or delete existing entries from the configuration file. The IPL_EDIT program can be used to modify existing entries. Also, IPL_TABLE.TXT is a type 4 file, therefore, EDIT/1000 can be used to modify this file. Caution must be exercised when using EDIT/1000 because the configuration file follows a very strict format. Refer to the LANVCP documentation in Chapter 12 of the *RTE-A System Generation and Installation Manual*, part no. 92077-90034, for the record format within IPL_TABLE.TXT.

Run IPL_BUILD. You will be prompted for information. Type the required information in between the "[ ]", left justified. The distance between the brackets is the maximum field length for the specified entry.

An example run of IPL_BUILD is as follows (note that two sets of entries have been entered for this run of IPL_BUILD):

```
CI> ipl_build
  Enter ipl table file name : <cr>

  With /files802/ipl_table.txt enter
 Name     [paint_shop         ]
 Number   [1      ]
 Address [080009000221]
 Download
     File [/gateway/ipl/paint_boot                              ]
 default flag [1]
       LU [36  ]
  continue ? y
 Name     [test_paint_shop    ]
 Number   [2   ]
 Address [080009000221]
 Download
     File [/gateway/ipl/test_system                             ]
 default flag [0]
       LU [36  ]
  continue ? n
```

The definition for each of the entries is as follows:

IPL Table File Name - The name of the configuration file. The default file is /FILES802/IPL_TABLE.TXT. IPL_BUILD can create or add to a file name other than IPL_TABLE.TXT, however, VCPMT always searches /FILES802/IPL_TABLE.TXT. Press <cr> to use this default file name.

Name – The name of your client node. Up to 17 characters can be specified. This name is used in the runstring for RMVCP.

Number – (Optonal) An integer number of the client node. Up to five characters can be specified. If the node number is not specified, the node number is zero. This number is used interchangeably with the node name in the RMVCP runstring.

Address – The 48-bit Station Address in Hex of the client's LAN card. This address was obtained in step 3 of the subsection "Client Addressing Configuration". Also, the last six Hex digits of the address are printed on the NOVRAM IC on the LAN card.

Download File – The file name of the memory-based system to be downloaded to the remote system (the client). Up to 63 characters can be specified. Specify the full path name of the system file; if the path name is not specified, RMVCP searches /FILES802 for the file.

Default flag – enter a 0 or 1.

1 = The system file in this set of entries is the default system file to be downloaded to the client in the case of a non-interactive download.

If the default flag is not set to 1 for any entry, then the default system file is determined to be the last entry in the configuration file.

0 = The system file in this set of entries is not the default system file (unless no set of entries have the default flag set, as described above).

Note that if more than one default flag is set (for a given Client Station Address), the last entry in the configuration file for that address with the default flag set will be downloaded. The use of the default flag allows you to reference a single physical node (Client Station Address) by two or more logical names (client names) in order to download two or more different system files.

LU – The user LU of the server's LAN card. This is the same LU number that was used in the "Linking the LANVCP Programs" subsection.

Continue – y = yes, add another set of entries; n = end program.

Client Hardware Configuration

Perform the following steps to correctly configure the client system hardware:

1. Set the U1 switch on the client's LAN card and install it into the client system. Note the following:

   - U1S1 should be CLOSED, if you want to enable VCP on the LAN card. In this way you can remotely access the client's VCP from the server system. Note that the client will not have a local VCP console. If you do not enable VCP on the client's LAN card, the client must then have a local VCP console from which the bootstring to boot over the LAN is entered. Or, if the client system is configured for autoboot from the LAN card, the download will occur on power-up of the client (with or without the client's LAN card enabled for VCP).

   - U1S3-S8 must be set to the select code of the client's LAN card. This must be set to 24 if you want the client system to perform an autoboot on power-up.

2. If you want the client system to perform an autoboot on power-up, set the BOOT SEL switches on the CPU to the following:

S1	S2	S3	S4	S5	S6	S7	S8
O	O	C	C	z	y	x	M

(Normally, S5 through S8 are closed.)

where C = closed
O = open
x = don't care
z = C, Normal Mode, BREAK Enabled on VCP console.
z = O, BREAK disabled. (VCP console won't halt CPU).
y = C, system console uses enq-ack handshake.
y = O, system console does not use enq-ack handshake.
M = C, disable auto-restart, battery backup not installed.
M = O, enable auto-restart, battery backup is installed.

Invoke VCP of the Client System

Invoke VCP and boot the client system in one of the following two ways:

1. Press BREAK on the client's system console. Enter the bootstring:

```
VCP> %bdsff00sc
```

OR

2. Run RMVCP to begin an interactive download session. Then, enter the /BREAK command and the bootstring as follows to boot your client system:

```
CI> rmvcp <client>
      .
      .
VCPMT> /b
      .
      .
VCPMT> %bdsff00sc
```

where <client> is the name of the client given during the IPL_BUILD process.

ff is the file number (in octal) from 0 to *77777*. If *ff* is not specified, *ff* defaults to 0 (and the "00" placeholder in the bootstring is not necessary, giving a bootstring of %bdssc). Also, *ff* defaults to 0 in the case of autoboot.

00 is a placeholder if *ff* is not zero.

sc is the select code of the client's LAN card.

Note that the *ff*00*sc* is not necessary during an interactive session, if the download file name is specified in the /FILES802/IPL_TABLE.TXT configuration file and the client's LAN card is at select code 24. (This is using the default of *ff*=0 and select code 24.)

Selecting the System File to Download

When a download is initiated (whether it is initiated interactively, from the client's VCP console, or upon autoboot), VCP on the client system sends a request out on the LAN for a particular file number. This file number is the file number that was specified in the bootstring.

If the file number is 0, VCPMT running on the server (who receives the request to download file number 0) performs a translation of file number 0. First, using the Station Address of the client who sent the download request, VCPMT checks to see if an interactive session has been started with RMVCP on the server system. If an interactive session has been started for that particular client (the client's name was given in the RMVCP runstring), VCPMT performs a search of the configuration file (/FILES802/IPL_TABLE.TXT). The system file in the configuration file associated with the client name will be the system file that is downloaded.

If VCPMT receives a request to download file number 0 and an interactive session has NOT been started (that is, the client has just initiated an autoboot), VCPMT will search the configuration file for the system file which has the default flag set. If more than one system file has the default flag set, the last entry with the default flag set will be the one that is downloaded. If no entries have the default flag set, the last entry in the configuration file will be the one that is downloaded.

If the file number is not 0, VCPMT translates the file number to form the file name P*fffff*. For example, the bootstring %bds70024, would initiate a download of system file P00007 from the server (client LAN select code 24). When the file number is non-zero and a P*fffff* file name is to be downloaded, VCPMT does not search the configuration file. VCPMT first searches the /FILES802 directory, then searches each FMGR cartridge on the server system until it finds the correct P*fffff* file to download.

Memory Dump Using LANVCP

The RMVCP program can also be used to take a memory dump of a remote node and store it into a file on the local node (server) for analysis. This is done by using the VCP command %wds from the RMVCP > prompt. When the %wds command is entered, the user is prompted for the size of the memory dump to be done and the file name of the memory dump file on the local node. A type 1 file of the correct size is created to hold the memory dump data. In addition, after the memory dump has completed, the user is prompted for 256-bytes worth of comments to be added (optionally) at the end of the memory dump file.

Realizing Future Computing Environments

James A. Baker
Hewlett-Packard Company
General Systems Division
3404 E. Harmony Road
Fort Collins, CO. 80525

Executive summary:

In the last 10 years, performance of computing hardware has improved nearly two orders of magnitude. Software on the other hand, has gained only modestly in harnessing the raw computing power to address the needs of users. This phenomenon is commonly described as the "software crisis". While no technological breakthroughs loom on the horizon, an organizational breakthrough could bring advanced technologies to users at an unprecedented rate. This organizational break-through is nothing more than the integration of the best available technologies in a vendor-neutral forum. The Open Software Foundation (OSF[1]) provides this forum and is just beginning to rapidly deliver leading-edge software technologies.

Through the OSF open process, leading technology can become standard faster than possible with current formal standards development. The open process also guarantees that the best technologies are examined and that selection is independent of a single vendor's interests. The rapid delivery of the best technology will motivate vendors and application developers to deliver solutions based on OSF software. Developers will divert their effort from the core software to easier, better integrated software solutions. The entire software industry will be accelerated as the amount of redundant effort squandered on core software development is applied to bringing computers "closer" to users. The OSF and the open process are the organizational breakthrough required to resolve the "software crisis".

Hewlett-Packard has long recognized the importance of open cooperation in setting and establishing industry standards. The involvement in the standards committees and the formation of the OSF give testament to HP's dedication to the open development process. HP is encouraged to see other computer vendors join in lending early support to the OSF/1 operating environment and looks to future OSF offerings to unite the computing industry while bringing superior functionality to all users.

Current Computing Environments

Computing environments today are inherently heterogeneous. This heterogeneity is a result of diverse, proprietary solutions to computing problems. While heterogeneity of computing environments is a consequence of innovation, it also strains the resources of users, application builders, and vendors. To utilize these computing environments effectively, the computing industry has sought standards to facilitate interoperability of hardware and software. Formal standards development however is a lengthy, slow process. The rate of innovation in computing is far outstripping the rate of standardization and consequently limiting the potential utilization of these advanced computing environments.

If computer vendors and solution creators were to agree on technology implementations soon after several diverse implementations were prototyped and evaluated, both groups could provide significantly more, advanced functionality. An early, co-lateral adoption of technologies would eliminate redundant efforts by software developer's and consequently allow software to leap-frog the current levels of functionality. Additionally, cooperative effort is likely to create synergy between the development teams yielding even better solutions.

Potential of Cooperation - SIGMA Project

The potential synergy of a cooperative effort is the impetus behind the Japanese Ministry of International Trade and Industry (MITI) forming the SIGMA consortium. SIGMA is a nation-wide effort to advance the state-of-the-art of software development in Japan. Specific objectives include improving software quality and productivity, and eliminating duplicated effort in software development. The entire Japanese software and hardware industry will benefit internationally by developing a common development toolset and methodology. The MITI recognized the unification of the software development resources as the competitive advantage for the entire country's information age competitiveness. While the benefits of the SIGMA project remain to be seen, clearly there is great potential for rapid advancement of the Japanese software industry given a common toolset and environment.

Open Software Foundation

Similar to the SIGMA group, the Open Software Foundation seeks to advance the software industry by providing the mechanism for synergy between software suppliers. OSF plays the role of an unbiased evaluator of competing technologies and as a facilitator for the open cooperation of competitors. Within the OSF open process, organizations that usually interact as competitors explore opportunities to leverage effort and standardize core system software. Exactly this sort of cooperative effort is required to ameliorate the "software crisis".

The crux of the open process is the mechanism used to specify, solicit, and select new technologies. The OSF staff members and special interest group's craft a request for technology (RFT) to specify required functionality. The RFT is an open invitation to the software industry to submit their diverse solutions to a specific problem. Selection from the plethora of technologies proposed is a mammoth task as there is seldom one technology superior to all others in all aspects. OSF staff members receive help from member company technologists and researchers to evaluate competing technologies. By using such a broad-base of RFT evaluation talent, the open process guarantees the needs of many different computing environments are considered during the selection process. The reasoning behind the technology selection decision is recorded and issued by the evaluation team for member inspection.

Request For Technologies - the core of the open process

The first of three RFT's issued from the OSF solicited recommendations for user interfaces. In a little over five months, the hybrid of two independent submissions, from DEC and HP, was selected and dubbed OSF/Motif. Based on the number of OSF/Motif licensees, the industry has acknowledged the power of the open process to deliver superior technologies at a rapid pace.

The second RFT requested distributed computing environments (DCE). Once again, many proposals originated from non-OSF member organizations. The best minds in industry and academia assembled to analyze and compare the wide spectrum of distributed computing technologies. The technology selected in the DCE RFT will provide the basis for interoperability of operating environments in today's heterogeneous computing installations.

OSF chose a lofty goal in the third RFT for an architecture neutral distribution format (ANDF). The ANDF RFT solicits functionality to allow solution software to be delivered in a hardware neutral format. ANDF technology would allow computer vendors to deploy solution-rich systems on the latest hardware platforms. Consequently, ANDF enables users to track the rapid hardware developments and still utilize all the applications in use today. Using ANDF, solution creators will provide a single version of an application for all hardware platforms. ANDF provides the potential to finally break the ties between hardware and application software.

In addition to the discrete technologies gathered by the RFT process, OSF will deliver a complete operating environment. The first release, called OSF/1, will incorporate the best technologies from several sources. The OSF/1 environment includes the best of System V, BSD 4.4, and Xenix application programming interfaces for optimal application portability. The core operating environment is built upon an implementation of the Mach version 2.5 operating system from Carnegie-Mellon University (CMU). The combination of widely used application programming interfaces, the best kernel foundation, and RFT technologies gives OSF/1 a distinct edge over all other available operating environments.

Benefits of OSF's Open Process

In the short-term, the OSF compliant software delivers a basis for future computing environments. OSF/1 delivers multi-processing capability, B1 level security, and a kernel architecture that facilitates easy migration from other environments. Many benefits of OSF are long-term benefits largely resulting from the synergy of development and elimination of redundant development efforts.

Of utmost concern for OSF/1 is the ease of user migration. If users of operating environments based on OSF/1 are not able to immediately become productive, acceptance of OSF/1 as a replacement for current operating environments will surely be slowed. Easy user migration is accomplished by providing all the applications currently in use today through a familiar user interface. Application availability is accomplished through source and binary portability for existing applications. The familiar user interface is provided by the industry standard OSF/Motif with it's Presentation Manager look and feel.

Ease of migration of applications is key to the success of OSF/1 as a viable operating environment. Application portability is accomplished on several different levels. Source-level portability is facilitated by the adherence to all dominant industry standards such as X/Open's XPG3[2] and the IEEE POSIX standards. The inclusion of de facto standards like BSD 4.4 compatibility and Xenix[3] application programming interfaces provides easy migration for a plethora of applications.

Binary-level portability provides the height of migration ease. OSF/1 provides a couple different approaches to binary compatibility. Using the OSF/1 functionality called transparent shared libraries, vendors can provide binary compatibility between existing proprietary systems and OSF/1 based environments. Where application binary interfaces exist for a given hardware platform, OSF/1 can conform. Both these approaches would allow existing applications to run, unchanged, on OSF/1. Binary compatibility is desirable in the short-term to assure application availability. In the long-term, applications will be attracted to port directly to the OSF operating environment due to superior functionality and longevity of an application's life through ANDF.

Today, OSF is delivering current versions of their development code so that vendors and solution creators can co-develop software in preparation for OSF/1 delivery in November 1990. Early availability of the software allows vendors to reduce the time to market of OSF/1-based products and allows OSF to substantially boost the quality of it's reference implementation through early feed-back. The snapshots allow vendors and solution suppliers to plan for the future and communicate the future to their customers. Running versions of the OSF/1 snapshots have been publically demonstrated at various trade shows including the summer Usenix conference (June 12-14, 1990 in Anaheim).

Hewlett-Packard (HP) is very enthused about the progress of OSF and the open process. By selectively implementing superior technologies, regardless of their origins, HP will preserve our customers current investment while moving to advanced technology base. Technology holds the answers to many computing problems but, HP's ultimate concern is for moving our installed base forward to new computing environments.

Migration Concerns

The ease by which users migrate to new computing environments is largely determined by the similarity of the old and new environments. OSF/Motif spans HP's current and planned environments, providing the bridge across a diverse product offering. Common user environments (NewWave) and applications round-out migration strategy assuring an easy transition for users to future environments.

HP has developed a two-pronged solution for the migration of applications to our future computing environment. First, application migration is facilitated through a common set of programming interfaces on all HP open systems. Second, the delivery of OSF technology based operating environment unifies HP open systems and provides a seamless migration.

A portability profile consisting of programming interfaces to the operating system, user interfaces, networking, and graphics across HP-UX, Domain/OS and OSF facilitates application migration. By aggressively implementing OSF technologies on Domain/OS and HP-UX, developers implement systems based on advanced technologies that will port effortlessly to future environments. Developers adhering to the HP portability profile will deliver long-living, portable applications.

HP plans on delivering an OSF/1-based environment as soon as possible after receiving the final source release from OSF. Utilizing OSF technology allows HP to merge HP-UX and Domain/OS environments onto an advanced software environment and provide a robust migration strategy to protect our customers investment in open systems.

Conclusions:

OSF's open process not only provides a cornucopia of new technologies, it also challenges computer vendors and solutions providers to compete and add-value in new ways. As the scope of OSF-provided computing environment increases, computer vendors will be forced to shift their software contribution from the core system software to higher levels of functionality. Solution creators will also face increasing competition as computer vendors start delivering functionality formerly only available from solution creators. While on first glance the OSF open process may signal the decline of the small, innovative software suppliers, just the opposite is true. Rapid standardization in operating environments will allow innovators to develop on a firm base and focus on their area of specialty rather than worry about application portability.

Future technologies that OSF is considering hold great promise for supporting current environments transparently. Thus, allowing users to take advantages of the latest in hardware developments while retaining their current computing environments. A system utilizing the technology present in the Mach 3.0 operating system from CMU, could concurrently support several operating environments on the same machine. This technology, called the micro-kernel, incorporates state-of-the-art functionality to facilitate trusted systems, multi-processing, specialized servers, and distributed computing. The micro-kernel is a departure from traditional monolithic operating systems and provides the basis for advanced computing environments for the coming decades.

Advanced technology aside, the OSF open process is yielding results far beyond those demonstrated by any other organization in computing history. Through the collective contribution of the entire software industry, the open process will allow the elimination of a significant amount of redundant effort by system and solution suppliers. At the same time, the open process poses a challenge to us all. The open process is a new way of doing business, a new source of technologies and above all, the open process is the organizational breakthrough required to alleviate the software crisis.

[1] OSF, OSF/1, and OSF/Motif are trademarks of the Open Software Foundation.

[2] X/Open and XPG3 are trademarks of the X/Open Company, Ltd.

[3] Xenix is a trademark of Microsoft Corporation.