M E M O

To: Ed McCreight

From: Peter Deutsch

Subject: The Alto software disaster

Date: February 23, 1974 [illegible]

Location: Palo Alto

Organization: PARC/CSL

Archive category: Alto (?)

Let us for the moment consider the Alto in its "state of nature", i.e. absolutely no software. We can see certain needs arising from the desires of various people to do things with this potentially interesting but momentarily unusuable machine:

A need for a reliable, permanent file system;

A need for an easy-to-use inter-Alto and extra-Alto communication mechanism;

A need for a multiprogramming operating system;

A need for a compiler, loader, and debugger for BCPL programs;

A need to combine (at least) BCPL and assembly languages;

A need for programmable overlays;

A need for display storage allocation;

A need for some kind of editor;

A need for an external file backup system;

And perhaps other needs which I am not aware of.

One way to fulfill these needs is to make them generally known and expect, encourage, or employ people to write each necessary piece of software individually. The predictable result would be a hodgepodge of conflicting conventions, duplication of effort, inability to put programs together into a cooperating or harmonious-appearing whole, and a lot of bad feeling. The alternative requires some leadership -- which can be democratic almost as easily as autocratic -- and some careful planning and coordination of the various pieces of essential utility software.

It is my belief that we are embarked on the first course; that there is already considerable bad feeling because of the delays and confusions surrounding the operating system and related software; and that there are problems just appearing on the horizon now which will lead to far more bad feeling in a year or so when we discover that none of our programs can talk to each other and that we have all had to invent adhoc mechanisms for things which could have been done well once. Here are some examples:

The file system, which is long overdue, has never been designed or documented on paper, so a number of people who wanted to use the Alto just went off and wrote their own (e.g. at least two different ones in Smalltalk) which are even less well documented or designed.

No thought has been given (as far as I can tell) to providing a facility which is present in Nova DOS but missing from TENEX and most other operating systems: the easy ability for a program to call on any function which is accessible from the executive language, and vice versa.

No thought has been given to multiprogramming (CPU scheduling, coordinated asynchronous processes) which is used heavily in Noxios and is often the most natural way to handle multiple I/O devices.

While the BCPL loader supports overlays, this facility is (as far as I know) not integrated with the operating system, so that (for example) the debugger must do its own overlaying, despite the fact the the o.s. must already have a facility for overlaying the exec with a subsystem. Nor do I believe that the BCPL debugger (such as it is) is capable of dealing with multiple-overlay programs, since it has no way of knowing what is in core. Nor has thought apparently been given to being able to load or debug assembly language code.

Aside from some routines which I wrote to allocate vertical space on the screen, there are no planned routines to help reduce the amount of core (and stolen cycles) used by the display. Numerous programs have already had to kludge their way around this, or else sacrifice the entire second 32K.

In view of the poor reliability of Diablo drives on the Nova, and the likelihood of bugs in the Alto file system, the Alto disk packs must be backed up by a reliable, permanent file storage medium. Likewise, an archiving medium is necessary since the Alto packs have limited capacity. Thacker's solution to this problem -- to provide one dual-drive Alto whose sole function is to provide a pack-copying service for users to utilize depending on their degree of paranoia -- is ludicrous. Just imagine trying to run that way on Maxc.

Despite some hardware progress on the Ethernet, no one appears to be thinking seriously about protocols (stream- vs. message-oriented, for example), operating system interface, whether some or all Altos will have "servers" and if so how they will be protected, etc.

None of these problems are extremely hard. Many of us at PARC, in CSL in fact, have seen them solved well or have learned from our experiences how to solve them reasonably well. It may even be the case that people are thinking about them and merely falling into the trap of believing that planning, discussion, and documentation can be left until after some "solution" is found. However, I believe strongly that globally good solutions will simply not be found in the absence of some direction and coordination. Of course, I am not volunteering either myself or you for this potentially rather time-consuming task. I do wish to point out that the last time I prophesied this particular kind of doom, resulting from bad planning and unwillingness to consider alternative courses of action, for the MPS project, I was pretty much right.

Inter-Office Memorandum

To	Alto Group	Date	March 5, 1974
From	Ed McCreight	Location	Palo Alto
Subject	Minutes of Alto Software Meeting on February 28	Organization	PARC/CSL

XEROX

As many of you know, the first meeting of the Alto Software and Chowder Society was held last Thursday (February 28, 1974). A number of topics related to the current state and future of Alto software were discussed.

The paucity of documentation in well structured and available form seemed to be the greatest irritant. To remedy this, Lampson agreed to provide a first version of the Alto Operating System Manual by the middle of March. This manual will consist of the Alto Operating System Design Notes augmented by lists of functions and descriptions of calling sequences, and by tables describing the formats of data structures maintained by the operating system.

What constitutes a releasable Alto Operating System? For the first release Deutsch listed a number of necessary features:

1. BCPL compiler
2. Assembler
3. BCPL loader
4. BCPL debugger
5. Source Language Editor
6. External file transfer mechanism (to/from Maxc)
7. Capabilities equivalent to those in Nova DOS command language interpreter
8. File access primitives equivalent to those in Nova DOS
9. Teletype simulation on Alto display
10. Keyboard handler with user access to interrupt routines
11. Direct disk access routines

There was general agreement that such a system was more or less minimal, that it would be useful, and that it would be by no means adequate for the long term. Metcalfe's list peers into the future:

1. DIAGNOSTICS!!!
2. Multiprogramming
3. Automatic (semi-automatic ?) backup facilities
4. Easy hard copy

Aside from lack of documentation, the major hang-up at the moment is in communications. For the very short term, backup can be handled by copying one's disk, and hard copy is more or less unavailable. Metcalfe assures us that if we can get parts, Ethernet interfaces will be available for production Altos almost as soon as they arrive, that at least one Nova interface will exist to connect the Ethernet to Maxc (perhaps indirectly via the MCA), and that interface software is being written. If this is not overly optimistic, then to construct elaborate kludges in the interim would be a misdirection of effort.

The general policy is that with the exception of things mentioned above, people will write their own subsystems. Lampson will endeavor to enhance the operating system itself to handle display management better, to allow dynamic loading and code motion, and eventually to include features to allow multiprogramming.

E. M. M.

Distribution

Gene McDaniel	David Liddle	Dan Swinehart
Butler Lampson	Jerry Elkind	Bob Metcalfe
Howard Sturgis	Bob Sproull	John Shoch
Chuck Thacker	Chuck Geschke	Larry Tesler
Jim Morris	Bill Duvall	Diana Merry
Peter Deutsch	Dan Ingalls	Alan Kay

Inter-Office Memorandum

TO: CSL, SSL

Date: May 24, 1974

From: J. Mitchell

Location: Xerox PARC

Subject: Summary of Programming Research meetings, May 14-17, 1974.

XEROX

Background:

During a consulting visit to Xerox PARC in January 1974, Alan Perlis suggested that there ought to be a long-term plan for Programming Research in CSL. He further suggested that the end goals of programming research should probably be driven by a view of Xerox Office Information Systems (OIS) of the future. Although a start was made at creating such a plan, day-to-day pressures and lack of the Perlis catalyst soon slackened any progress.

Dr. Perlis again visited CSL from Monday, May 13 to Friday, May 17 and J. Elkind suggested that programming researchers and Perlis ought to have a series of meetings to revive the notion of a plan. This report is a condensation of those meetings, which took place May 14-17 each morning.

The following people attended at least one of the meetings:

P. Deutsch, J. Elkind, E. Fiala, C. Geschke, D. Liddle, E. McCreight, J. Mitchell, J. Morris, J. Rulifson, E. Satterthwaite, J. Shoch, R. Sproull, H. Sturgis, R. Sweet, R. Taylor, L. Tesler, C. Thacker, B. Wegbreit, and A. Perlis.

Briefly, the first meeting, on Tuesday, tended to be concerned with programming research, the problems of producing software in CSL, and possible future forms of an OIS. On the second day, Perlis suggested that a small number of languages of expression should be used and be able to be depended upon for most of CSL software. This rapidly focussed on what collection of such "gold coins" of exchange should be provided for Altos. On Thursday this problem was the exclusive topic: MPS on Alto, byte-LISP and the Alto operating system (in some form) were generally proposed as the initial set of gold coins. How to make them materialize as such was much discussed but only partly resolved. It was proposed by Elkind that a summary of the meetings be made generally available and then that a small group decide the design and implementation issues of a set of gold coins by June 15, 1974. The final meeting on Friday reverted to a discussion of programming research goals.

Tuesday, May 14:

Mitchell stated that the purpose of the meeting was to understand the long range aims of programming research. He suggested that this be discussed in the context of future OIS possibilities. Broad goals included producing and maintaining software systems, language and system facilities to aid programmers, software organizations for network environments, etc. In particular, the manner by which CSL produces software for public use should be a subject of scrutiny.

This led to a general discussion of problems with CSL's current schema for software production: namely, someone, after generating a high-level specification for a task to be programmed, gives the problem to a single person who is then expected to go away and do it. Those who accept such assignments are generally viewed as "second-class citizens", and the only questions generally asked of them is when it will be done. This feedback loop does not seem to be tight enough or to carry enough information about difficulties and progress for monitoring purposes. Perlis said that those who contribute to the public good in this way should really be viewed as very high class citizens and supported appropriately.

The discussion was diverted back to the original, stated purpose of the meeting. The outcome of this was a long list of possible OIS features and uncertainties, which did not seem to lead anywhere.

Wednesday, May 15:

Perlis began by saying that CSL (we) should, in part, view ourselves as professionals for software design, production and maintenance. Second, any system produced in CSL for public use should have a responsible set of parents who will care for and nuture it. Third, since the CSL environment should reflect our concern for software, we ourselves must act responsibly, and the act of providing service must be a subject of research.

More specifically, we need a few agreed upon languages for the expression of problem solutions which are implemented and maintained on Altos. Together with the Alto operating system, these language systems must be viewed as "gold coins" which may be depended upon as a solid basis for other software and research ideas. A gold coin was defined to be a software facility that is fully supported in the sense that bugs will be fixed, that improvements will be made so that it is efficient for CSL research, and that user-level documentation will be provided and updated periodically. Perlis further stated that it was important to establish firmly a set of gold coins, even if they were not the ultimate, best systems. (The phrase, attributed by Perlis to Wittgenstein: "The best is the enemy of the good;" was heavily referenced).

As a starting point, he stated that Lisp and MPS on Alto should be the main gold coins. It was pointed out by Elkind that this certainly also made the

Alto operating system a gold coin. Perlis agreed and said that therefore the needs of Lisp and MPS should in large part determine the design of the Alto operating system. Deutsch stated that he though that this was an important point and that he (as a responsible Lisp parent) would like to see a smaller "kernel" set of facilities which would make an Alto into a nice virtual machine for Lisp. There was much discussion on the differences between such a kernel and a more general operating system, with little resolution.

Sproull suggested that programs written in gold coinage should be able to communicate (and to some extent coexist) in an Alto so that programs and subsystems could be generally used without having to write versions in each of Lisp and MPS. This was generally accepted, although there was no agreement on the degree of communication which would or should be possible. Mitchell suggested that laguage-to-language communication might well require some minimal virtual memory or swapping capability acceptable to all the gold coins. Sproull suggested that a communication protocol need not necessarily involve operating system code unless the communication required Ether or Sig Network transmissions. A sequence of straw votes revealed no dissent from the proposals made by Perlis.

Thursday, May 16:

Perlis asked whether there should be other gold coins besides Lisp and MPS. Discussion with J. Shoch effectively ruled out SmallTalk as a candidate. Perlis suggested that an APL system for Alto also be considered for gold coinage; the general consensus was that it not be considered as one of the initial set, but possibly be added later.

At Elkinds suggestion, a list of necessary Alto support software was made. This was quickly pruned to the following set:

- Lisp and MPS
  - a primitive assembly language facility in Lisp and MPS
  - loaders and debuggers within Lisp and MPS
- microcode assembler, loader, and debugger facilities
- kernel operating system to support Lisp and MPS
- various utility routines (FTP, printing, etc.)

It was pointed out that this list represented some final state of gold coinage to which CSL should aim, and other facilities such as BCPL have to be supported on an interim basis until not needed. This list coincides in large part with that in McCreight's memo "Minutes of Alto Software Meeting on February 28":

- BCPL compiler, loader and debugger
- minimal Assembler
- source language editor

- external file transfer mechanism
- command language interpreter (like Nova DOS)
- file access primitives
- teletype simulation on Alto display
- keyboard handler
- direct disk access routines

Perlis suggested that comparing two such laundry lists was a technical problem which should handled by a small group of people responsible for gold coins. Elkind responded with two proposals:

(1) Mitchell should write a summary of this series of meetings for general distribution;

(2) a group including McCreight, Geschke, Lampson, Elkind, Deutsch, Mitchell and representatives from SSL should review the Alto gold coin proposal and produce an iteration on it as well as a proposal for implemention by June 25, 1974.

Taylor pointed out that one of the reasons for the current "level" of the Alto O/S was the amount of support which could be given for it and that changing that level of support would present problems. Perlis pointed out that a major problem with public systems is the proverbial "last inch": the part after the interesting portion of the system is done, but which is necessary to make it usable. He and Simonyi maintained that, as research types, we are generally ill-suited to making the last inch and that we should recognize this in deciding what level of support is needed for producing usable software. There is some chance that Mr. Simonyi was attempting to whet peoples' appetites for his seminar on "the Software Factory" later that same day.

Friday, May 17:

Elkind alleged that the programming researchers in CSL want to learn to make software easier to build and more reliable. He asked Perlis what he thought CSL should be doing in programming research besides. Perlis replied with a Perlisism:"computers are always capable of more than we know how to make them do." Therefore, finding notations for reducing the complexity of what programmers do should also be a goal of CSL, as should different ways of modelling and thinking about programming. He claimed further that it is generally more fruitful to work from problems to understanding that vice versa.

Mitchell stated that a part of the research goals of CSL should be to make any knowledge gained by attacking such problems transferrable. In general, we do not disseminate very well the design reasons and internal structure of systems we build (with the notable exception of hardware).

At this point, Perlis described a set of criteria due to Cobham of IBM for deciding what IBM should do in the area of Artificial Intelligence. He suggested that these criteria might also be applicable to what CSL and SSL do. Cobham's overall suggestion was that IBM should not attack large, open-ended, AI problems such as robotics, but concentrate instead on heuristic solutions to specific problems. Those problems and solutions should satisfy the following criteria:

(1) there should not already exist a good way of solving the problem;

(2) there should be a large body of experts in the field of interest;

(3) solutions to the problem should be testable to determine how well they solve it;

(4) the experts in the area should perceive that the solution to the problem has real value to them.

Elkind added another criterion to this list in the case of products for more general use: they should not force their users into modes of operating shich differ from their normal ways of thinking about the problem.

As a closing comment, Perlis reiterated that programming research should be fairly self-reflective about what it does when producing software and tools for itself and others.

Inter-Office Memorandum

To: CSL, Office Communication Group

Date: May 30, 1974

From: P. Deutsch (as scribe)

Location: Palo Alto

Subject: Gold Coin meeting

Organization: PARC/CSL

File: MAY29
Archive category: Programming Research

XEROX

The following are the minutes of the meeting held May 29, 1974 at Butler's house. Attendees were: Butler Lampson, Chuck Geschke, Jerry Elkind, Charles Simonyi, Ed McCreight, Jim Mitchell, and Peter Deutsch. *In return for accepting the responsibility for producing the minutes, I have taken the privilege of inserting my own after-the-fact comments (in italics).*

1. Preliminaries

Some brief clarifications were voiced on the subject of Jerry's "Gold Coiners" memo, which served as the agenda for the meeting. Under "Assignment to Lisp and Mesa groups to produce implementation plans for Alto" we agreed to include the operating system as well. We agreed that the "Standards for Gold Coins" subheading essentially meant to define the phrase "Gold Coin". "Hardware" signified our willingness to contemplate significant add-ons to the Alto, but not rebuilding from scratch.

2. Hardware

The first topic was what kind of hardware support Gold Coins should require, beyond the standard Alto (1K PROM, 64K core, small disk). The discussion dealt entirely with the issue of RAMs vs. PROMs. If G.C. microcode is relatively static, it is all right to use PROMs, since they only cost 1/7 as much as RAMs (at the moment; this may be overtaken by technology changes), so it is just a question of controlling the release rate for new versions of the microcode. However, Lisp and Mesa microcode will probably not fit into a 1K PROM together. For experimentation, CSL will probably want RAMs in all its Altos eventually; an Alto can also accommodate an additional 2K of PROM in its current packaging, for a total of 3K, in lieu of 1K of RAM. Conclusions: the operating system should not require any microcode support beyond the standard 1K (of which about 300 words is still free); each of Lisp and Mesa should fit in an additional 1K; neither system should require dynamic swapping of microcode for its own purposes. This makes the following configurations possible: an additional 1K PROM, committed to one of the two systems (good for SSL, which will probably only want Mesa); a 1K RAM, loadable for either of the two systems (probably standard for CSL); an additional 2K PROM, with both systems. We agreed that there was a cost tradeoff to consider between the more expensive RAM and the number of PROMs required to cope with releases of Lisp or Mesa.

3. Documentation of G.C.s

Next we considered documentation standards for Gold Coins. We agreed that reference manuals, though onerous to write, were absolutely essential: we do not believe in "documentation by word of mouth". (The Interlisp manual, and the Tenex JSYS manual, respectively represent upper and lower bounds on the expected scale of documentation.) We also agreed that true introductory manuals are far more difficult to write, and that PARC people, being relatively sophisticated, could be expected to learn all they wanted from a reference manual and asking questions.

There was less agreement on the subject of how to document the workings of the programs themselves. We agreed that traditional "internals manuals" were uninstructive and not worthwhile. BWL and LPD favored self-documenting code, and nothing else. BWL and CS suggested that meta-programs could form part of this self-documentation, once we understood what they were. JGM noted that it was important to set down the reasoning and experience that led to each important design decision, as well as a description of the result. (This was related to Perlis' comment about being reflective as we write our programs.) JIE ended the discussion by stating that Jim Morris was concerned specifically with program clarity and readability as it affected maintenance, and that we should therefore push the research problem off on him and muddle along as usual in the meantime. *I have some qualms about this. I think we should make a deliberate effort to get people to put comments into their code, especially the data structures. It's a lot easier to read code with comments only on the data than with comments only on the procedures.*

4. Improvements and maintenance

On the subject of maintenance of G.C.s, we agreed that release of a G.C. obligated the releaser (or appropriate responsible party) to maintain said G.C., meaning to fix bugs in a timely manner. (There was a digression on a bounty system for users who report bugs.) It was pointed out that the requirements for maintenance are conditioned by the fluidity of the system: Interlisp, which is continuously evolving, always has a few minor bugs, whereas BCPL no longer requires any attention at all. Since Alto Lisp and Mesa will be closer to the Interlisp model (*my impression: I think this was a consensus*), we can expect ongoing maintenance requirements.

Improvements were more controversial. We agreed that we needed a mechanism for deciding whether a new feature or change should be made: it must be possible both to restrain overzealous implementors and to satisfy genuinely needy users. This is tied up with the question of what level of support (in terms of responding to user-requested improvements *I think*) a Gold Coin commitment implies. JIE pointed out that it is essentially a question of people resources: by adding more people to a group, we can have as high a level of feature development and of system reliability as we want. (As a counter-example, there is WT, who is not replaceable nor easily augmenttable vis-a-vis Interlisp.) BWL attempted to sum up the problems by asserting that there are three ways to get in trouble: constructing a basically unreliable system, like PUB (a "gold coin with a brass heart"); letting implementors have too free a hand to add features; and making it too hard for users to get really vital changes made. We agreed that we didn't know what scale of ongoing change would actually be forced on us by our research needs. In particular, we did not have a clear idea of what would be required to satisfy POLOS needs. *It was a mistake not to invite someone from the Office Communications group to the meeting.*

Wegbreit has expressed severe doubts that viewing the operating system in this way will do anything to alleviate the interconnection problems that seem to make all operating systems so hard to change.

*** At this point LPD had to leave; the remainder of these minutes are a reconstruction from JGM's notes. ***

Most of the remaining discussion was taken up with memory management and multiprogramming issues. CS proposed that base-bounds pairs be implemented: these are good for protection, and have some value for memory shuffling: the o.s. could ask the program in a given partition for additional space, which the program could give back from either end of the partition. BWL didn't like this proposal, nor JGM's subsequent proposal for 940-style (Tenex-style) paging. It was concluded that the only real value of such schemes, given that the Alto was committed to a 64K virtual space (the same size as real core), was for protection between co-resident subsystems, valuable only for debugging or in cases of mutual suspicion. On the general issue of protection, BWL pointed out that the Alto environment for a <u>single</u> program is no more dangerous than that of a user program on Maxc.

This discussion led into a more general discussion of multiprogramming. BWL was against it, JGM for it. *There are two issues involved. One is multiple processes, scheduling, wakeups, etc. Some provision for this is absolutely essential, for real-time experimentation, background services, and the needs of Pogos. The other is the question of mutual protection of suspicious subsystems. It is the latter that appears to be the rat's nest, and I agree with BWL that we should not get too far into it. Since the line between "user" and "system" is fairly fuzzy on Alto, we can afford to rely on a certain amount of cooperation or at least toleration between co-resident programs. Jim Morris also pointed out subsequent to the meeting that the Alto already suffers from lack of protection in one nasty respect: if a program crashes, it is very likely to take the display with it, so there is no record whatever of what was happening other than the contents of core.*

It was agreed that the things on which we must agree in the design of a viable operating system for language system support include: a file system; a scheme for partitioning and allocating core; a communication protocol *(does this mean Ethernet, inter-language, or both?)*; a set of multiprogramming conventions including memory protection *and a process system*. It was suggested that a base-bounds scheme could accomplish the second and fourth of these functions. The idea of supporting a virtual memory system was deliberately left off the list. *There is some chance that the Lisp virtual memory (2↑24 32-bit words) will be acceptable for Mesa, in which case some of its mechanics could be delegated to the operating system.*

BWL asserted that he would undertake the completion, maintenance, and documentation of Gene's embryo operating system. In particular, he said he would: keep the file, directory, and stream systems essentially as they were; add facilities for swapping code and allocating memory; add some display allocation routines; and provide a stream interface for the Ethernet. *Maybe the Diablo printer too?* The memory allocator would include both a standard allocator in which a block's location was fixed, and a "floating" allocator which would allow blocks to have associated fixup procedures, to be called when the block was moved.

This ended the meeting. JGM added the following in his notes (private thoughts): he wants to write down some of his own thoughts on operating systems, so that BWL is not solely responsible for the resulting design -- the issue is not BWL's competence, but rather the historical observation that systems almost invariably turn out better if a small number of people (plural) contribute to the design, and the pragmatic observation that BWL may not be able to represent to himself the multifarious interests and needs of the potential users of the o.s., namely the Lisp system, the Mesa system, and Pogos.

We then addressed the question of what commitment we were making to satisfying the needs of OCG. It was observed that with the Alto hardware, which we agreed was itself a G.C., they came in after the planning and design had been done, and were willing to accept our posture that we wouldn't support anything beyond a standard system. With Mesa, they are likely to have needs for a process system (for Pogos) beyond what we see ourselves needing at the moment. We agreed that having two versions of Mesa would be very bad. On the other hand, we speculated that their (perceived) need to squeeze the last 10-15% of performance out of the machine might lead them to be dissatisfied with mechanisms that we found adequate. We agreed that resolving such things would always be a matter of negotiation between the groups, in which we would have a basically stronger position since we were the primary originators and maintainers of the programs. *I am very uncomfortable with this approach. I think if we are serious about Mesa as a real tool for building real systems, we will have to get OCG people into the design process as soon as possible, and make some kind of commitment about listening to them in the future. I don't have a good feeling for what that level of commitment should be -- we should have dealt with it more specifically.*

5. Intercommunication

There was some concern about the ability of Lisp and Mesa programs to talk to each other. In the long run (>1 year), we will put Lisp on top of Mesa: JGM assured us that the Mesa space overhead was small enough to make this feasible. The problem of differing calling conventions for Lisp, Mesa, and possibly the operating system was deemed solvable *by a kludge*. JGM suggested that whatever communication mechanism was adopted should allow interposing an Ethernet link between the two parties. We agreed that this required restricting passed data to scalars, since we didn't understand how to pass pointers between machines. Furthermore, there was some question as to whether Lisp and Mesa would even use compatible virtual memory schemes. It was decided to push this (technical) question off onto a separate meeting of LPD and JGM.

CS proposed a different approach. In his view, the operating system provides lots of abstractions through standard services: these are precisely the useful language-independent abstractions, and therefore a natural medium for inter-language communication. This means, for example, that it would be possible to provide a communication mechanism (possibly even encompassing JGM's Ethernet suggestion) that looked like an o.s. call to both parties.

6. Operating system

CS proposed that one put as much as possible into the operating system which will support Lisp and Mesa. *I originally opposed this idea on two grounds: one, that there is a severe risk of creating a monster; the other, that the core cost was too high. JGM assured me that the core cost for an unused feature could be reduced to a few words by use of a code swapper, which itself could be very small.* After some discussion of virtual memory facilities, we agreed on the principle that an o.s. should provide things thought of as "services" but not "decisions", e.g. it might provide the mechanics of paging but not the page replacement algorithm (about which the language systems should have a better idea). We noted that this resembled the idea LPD had pushed at the time of Perlis' visit, namely the image of an operating system as a small kernel and a large cloud (husk?) of subroutines surrounding it, although the contents of the kernel would in this case just be the code swapper. *Ben*

M E M O

To: CSL and SSL

From: Howard Sturgis

Subject: Gold Coin Meeting, June 7, 1974

Date: June 11, 1974

Location: Palo Alto

Organization: PARC/CSL

I) Preliminaries

Transcription of notes taken at a GOLD COIN meeting of friday, june 7, 1974.

These notes will be written as a series of paragraphs, with no further attempt at organization. My after the fact remarks will be in square brackets.

The meeting was at Butler Lampsons house and was attended by:

Butler Lampson
Jerry Elkind
Peter Deutsch
Jim Mitchell
Chuck Geschke
Charles Simonyi
Ed McCreight
Howard Sturgis
Bill Duvall

Howard Sturgis was choosen as secetrary on the grounds that he was one of the two new membersof the discussion, and was the only self invited new member.

II) PURPOSE

Jerry Elkind stated that the purpose of these meetings was essentailly that of a subcommittee. We were to form a plan of attack on the gold coin problem and then put the results of the plan on the floor for discussion. (By Full CSl etc.) There will be at most one more meeting of this subcommittee before reporting to all of CSL.

III) VIRTUAL MEMORY

Next we launched into a long discussion of virtual memory. [Two issues seemed paramount: is it possible to have a common scheme for all ALTO languages, and can we decide the issue now.]

Do we have to go through the map every time? Not in LISP proposal. Not in HES-BCPL un written proposal. Under Charles proposal, all memory referances go through the map, but an associatiove memory is included so that repeated referancecs to the same page do not have to be mapped each time.

JIE: the problem: Charles, Howard, Peter, Jim etc have to get together and see if there is an underlying virtual memory structure and mechanism that all can use.

BWL:why we want compatibility:

a) So that when one gold coin passes an address to another it can be interpreted.

b) So that implimentation is done only once.

c) BWL re-interprets Charles 'WHY' to: EAch seperate process runs inside of protection. i.e. Seperate programs run without conflict.

LPD: All the language systems need a large virtual address space. Where should it be provided? (i.e. who provides it?)

a) In the language system
b) in the machine
c) in the operating system.

BWL: 1)MESA needs virtual memory. 2) We need some statistics on the pattern of memory referances in MESA. 3) design the virtual memory on the basis of the statistics. Makes no differance what form the virtual memory takes, it will provide protection.

JIE: Those who are involved should design a "good-common virtual memory".

BWL: advantages of common, as opposed to individual, virtual memory schemes: 1) common implimentation (need only be done once, most advantageous if hardware needed) 2) can pass addresses between the gold-coins.

It is possible to concieve of schemes that use different schemes for the different languages but the same hardware and same addressing, but implimented differently.

IV) MultiProgramming

Discussion of Charles "2nd Alto MultiProgamming Trial Balloon" [I asked for , but did not recieve, a copy.] [My memory and notes for this discussion are very confused.] [first some general questions were asked by assorted individuals.] Should there be software interrupts? Hardware interrupts are converted into event channels. Should then event channels be converted back into interrupts? Is a process expensive or cheap? (This will determine their use).

HES: MESA modules look like processes, Ports look like event channels. Jim remarked that that was a misconception. [I will have to learn more about MESA].

Charles: Processes and event channels should be expensive. Bill Duvall: Processes and event channels should be cheap.

[now comes a gap in my notes labeled "very long discussion", seemed to include:] Where do interrupts fit into language structure. (by interrupts here, we mean pre-emption). How does this multiprogramming proposal interract with virtual memory and address spaces?

V) Gold Coin Utilities and Library Functions

Symonyi and McCreight produced a document "Proposed golden utilitis and Library Functions". This document proposed a list of 10 utilities and 10 library packages which should be considered for gold coinage.This provoked an item by item discussion which will lead to a new version of the document to appear subsequently. [The following paragraphs reflect those topics which I succedded in writing down.]

Where should the executive lie? In a display editor, in Lisp, or in MESA? An executive is a progrmming system, hence should be imbedded in an existing language.

Debuggers. Should be a machine language debugger (for the machine maintainers, at least). (Even though the machine is a gold coin it does not follow that all tools required to maintain the machine are gold coins.) There should be a debugger based in each language.

BCPL. BCPL is a lead coin, not quite a gold coin, but needed for now. Hence any virtual memory system should permit BCPL (as presently compiled) to run.

Peter: send msg should be a utility, while intermachine communication should be a library routine, used by send msg and other utilities. Sending a msg and sending a file are different. Should be relatively easy for any program to send a file to another machine, But sending a message will be a utility since it will have a lot of associated baggage.

Bill Duvall: Proposes as an additional gold coin a file maintaining facility. It will some how assure that all users have current versions of a given set of files.

Discussion that "Gold Coinlets" (utilities, not language systems) should be self documenting. This is to replace a stack of documents.

Ed McCreight will produce an edited version of "McCreight and Simonyi" At the least, it will have machine language debugger removed from gold coin status.

Geschke: We should not depend on MAXC in order to run. MAXC (or the connection to MAXC) may be down. This was in response to discussion about file storage and time of day service to be performed by MAXC.

Some discussion about strings: should their representation be common among the gold coin languages, or language dependent?

Floating point numbers: Make a new library function for all floating point number stuff. i.e. input output conversion, math library etc.

Decision: all languages will use IBM 360 floating point representation. Thus a library of good math routines already exists, and all we have to do is transliterate it. This is at the cost of a slight loss of precision due to the HEX exponents.

Peter Streams:

1) at present back end of a stream looks different from the front end. Want a co-routine structure so that stream code can have it's state automatically saved.

2) The current Song and Dance about stream storage allocation is too complicated. (BWL: it will be improved) Want to be able to supply stream space explicitly to the operating system. Thus Peter can control the usage of storage space as his needs dynamically fluctuate.

3) How do streams interact with module coupling facilities in MESA?

Howard: We want multiple directories supported.

Peter: Computation and IO must overlap in the new operating system. At the least, disk transfers and computation should overlap.

VI) FOR NEXT MEETING

For the next meeting: Subgroups will discuss virtual memory stuff. Some rough schedule will be produced for MESA, LISP and the operating system. (BWL: the operating system changes will be incremental, rather than a re-write.)

Bill Duvall will produce a schedule for his project to see how it meshes with that of the gold coins.

Next Meeting: Monday june 17.

MEMO

To: CSL, OCG

From: P. Deutsch

Subject: Proposal for a partially automatic source file bookkeeper

Date: June 16, 1974

Location: Palo Alto

Organization: PARC/CSL

File: SFB
Archive category: Programming research

Bill Duvall mentioned at the second Gold Coin meeting that he would like to see some aids for keeping track of the migration and alteration of the multifarious source files that compose the Office System software. Similarly, I have wanted such aids for some time, both for my own work with large, complex programs and to aid in the maintenance of Interlisp. It seems to me that any system we contemplate along these lines should have the following properties:

It should be fairly non-language-specific. By this I mean that it should be possible to extend it to cover new languages by writing a modest amount of new code, as opposed to major alterations in its structure.

It should be network-oriented. By this I mean that it should not assume that it has access to every file of interest to it at every stage of that file's evolution.

It should be free-standing. By this I mean that its use should not require alterations in existing language processors, editors, file systems, etc. This point is likely to be controversial, since much of the success of Interlisp's tools, for example, arises from their close integration with the language system.

It may be that these properties are overly constraining -- that they put unacceptable limits on the future evolution of the system we build initially. I would like to hear discussion of this issue. However, I believe an initial system can be constructed which satisfies the above criteria in the following specific ways:

It can handle BCPL, ASM, Interlisp, and Mesa programs. In certain cases, it does require the insertion of extra comments in the source code about the interrelationships of files.

It can deal with programs that are shipped off to Novas, Altos, or other ARPANET machines. However, it assumes that the "home location" for programs is a Tenex system on which it runs.

It does not require alterations to any existing programs. Its only interface to the rest of the world is through the terminal and through standard Tenex source files.

1. Abstractions

Taking a leaf from Charles Simonyi's practicum, I will enumerate the abstractions which the proposed system deals with.

The most important concept, of course, is the *file*. Files come in two or more flavors: *source* files, and *derived* files (.RB, .BR, .COM, .SV, .SAV, etc.) that are generated from source files or intermediate derived files by a variety of processors (ASM, BCPL, Interlisp compiler, BLDR, etc.). The proposed system does not necessarily know how derived files are constructed, but it does need to know how to correlate derived files with the source files that produced them, and vice versa. Source files also carry, permanently, a designation of their *syntax class*, i.e. what language they are written in. In addition to their *content*, files carry a *write date* and an *author* (the name of the person who last modified them).

Internally, source files carry several different kinds of information of use to the system. The programs on a file are divided into *units*. These may correspond to procedures, groups of procedures, etc.: it is with units that the system associates information about editing history and responsibility. For each syntax class, the system knows how to identify the unit boundaries in a source file of that class. Some of the program may not fit naturally into a unit, for example the comments at the head of a file. This is also acceptable. In either unit or non-unit portions of the program, there may be *external references* to units in other files. (By definition, a unit is the only thing that can be referenced from another file -- maybe?). In BCPL, these are the "get" statements. In ASM, comments are required to identify the files in which .EXT names are defined. In Interlisp, comments are required to specify the files which must be loaded for a given file to run. Mesa has something similar to a "get" statement, but additional comments may be required. Finally, the system itself adds comments at various points within the source file to record its history in human-readable form, and at the end of the file for its own purposes. ?

2. Operation

The proposed system could operate in either "executive" or "worker" mode. Executive mode would be rather like DEC's CCL: there would be a command language including operations like edit, compile, send/receive, query, etc., and the system would call in the appropriate programs to do the work it could not do itself. Worker mode implies running the system separately from the editing, compiling, etc. operations; in this mode the system might have to go to more trouble to determine what had happened "in its absence".

We now consider what the system should be able to do. The particular set of user-initiated operations which are of interest to the system are as follows:

Edit a source file;
Process a source file or intermediate file to yield another derived file;
Make a copy of a file, possibly involving another machine or directory;
Create a new file, or rename, split, or merge existing files;
Release an executable file to a user community.

To be more precise, these are the user-initiated operations which can take place outside the proposed system. It is likely that the system will not be able to reconstruct the necessary information if the user performs arbitrary combinations of these operations outside its purview; further thought is needed to determine what combination of requiring certain operations to be performed from within the system, on one hand, and providing incomplete service, on the other, is appropriate and feasible.

As should be clear by now, the system operates largely on the basis of source comparisons between versions of files, comparison of write dates, and history information recorded in source files in the form of comments. It should be possible to provide the following services:

Determine which processors need to be run, on the basis of write dates of derived vs. source files (CCL exists essentially for this purpose);

Determine where the current versions of files reside, and collect them together;

Provide multi-way comparisons to aid in merging independent changes, or prevent such situations from arising;

Provide responsibility information at the program unit level, in case of user complaints;

Retain information required to identify the consistent set of source files which produced a given release of a program.

I would very much like to see further discussion of these suggestions. I think it would not be too difficult to implement a system with all the aforementioned properties. I would propose that it be written in Interlisp -- this makes it somewhat less accessible to Nova users, but it will be available on Alto. Alternatively, the Mesa group may have plans along these lines.

Inter-Office Memorandum

To:	CSL, SSL	Date:	June 18, 1974
From:	Chuck Geschke & Jerry Elkind	Loc.:	Palo Alto
Subject:	Standards for Gold Coins	Org.:	PARC/CSL

XEROX

The purposes of this memo are (1) to define what we mean by the term "Gold Coin," (2) to enumerate the principal Gold Coins that we have identified to date, (3) to set forth some of the properties that characterize Gold Coins and (4) to identify some problems with the administration of the proposed "currency of the realm."

Definition

A Gold Coin is a dependable, efficient software or hardware facility which services a wide spectrum of Alto-based CSL/SSL computer and systems research activities. Gold coins will be fully supported in the sense that bugs will be repaired, improvements will be made, user-level documentation will be provided, compatibility will be maintained, etc. The set of Gold Coins will cover most of our common needs for tools.

Enumeration

The following is our list of specific facilities that will be included in the Gold Coin set initially:

- Alto
- Ethernet
- Operating System Kernel
- Operating System Utilities
- Mesa
- Interlisp
- Office Services

The Office Services are the various systems being developed by SSL as part of their research on office communications that will be in widespread use at PARC.

Since it will take some time to achieve a coherent and complete set of Gold Coins, some interim facilities like BCPL will be supported. It is likely that in the future other facilities will be proposed for inclusion in the set of Gold Coins. The discussion below provides some possible criteria against which these proposals can be evaluated.

Properties of Gold Coins

We have chosen to classify the properties that characterize a Gold Coin under four categories: completeness, research relevancy, longevity, and general usefulness.

Completeness

The set of Gold Coins will constitute the common base of facilities and tools upon which much of CSL's and SSL's research will depend. The set must be made complete enough to support all of our principal research projects by providing those facilities that are used in common. Our initial notions of what constitutes an adequate set of facilities is derived from our present computing environment. The best examples of Gold Coin-like facilities are Tenex and Interlisp. The Alto Gold Coin set must cover our needs at least as well as the Tenex system and subsystems do.

Research Relevancy[1]

Creating and maintaining Gold Coins will require a large commitment of time and energy over an extended period. It is important therefore that Gold Coins for the most part be in the mainstream of our research and not be viewed as extraneous and offensive social obligations. Indeed, as Perlis has argued, one of our principal research objectives should be to learn how to build and maintain systems of Gold Coin quality. But more than that, the principal coins should be congruent with principal research projects in the way that Mesa and Interlisp are. Of course, some Gold Coin-like facilities will not have this intrinsic interest to us, but that need not be a serious problem provided that most of the treasury is filled with gems of research quality. But beware of error 33.

Longevity

If Gold Coins are to provide the bases for ongoing research activities there must be reasonable guarantees that the Gold Coins themselves will be continuously supported for a long time into the future. In addition the Gold Coins must be capable of evolution, both because they or the problems they address are objects of research and because the requirements of the user community will change over time. Thus the keepers of Gold Coins have an obligation to maintain compatibility between new and old versions of their system, or otherwise to help users make a transition between systems.

General Usefulness

We will not attempt to maintain as Gold Coins those facilities that are used solely by individual research projects. Such facilities are the responsibility of those projects. Gold Coin status will be reserved for facilities which are of more general usefulness.

The fact that Gold Coins will be used widely and that users will not attempt individually to maintain and develop them, imposes some significant responsibility upon the keepers. We should insist on high standards of reliability, repair, and responsiveness to evolving demands. The keepers must furnish documentation at a level appropriate to the average user of the facility, typically at the reference manual level not the primer level. Care must be exercised to insure reasonable compatibility through successive releases and documentation must be kept current. This level of support requires significant resources because the tasks are large and will continue in time. Succession of responsibility must be maintained over the life-time of the facility.

Mechanics of Administration

There seem to be three major administrative issues that must be resolved:

(1) deciding whether or not to embark on a new Gold Coin development venture.

(2) certifying that particular facility satisfies the criteria for Gold Coins, and

(3) managing the development and long-term maintenance of a Gold Coin in a way that is responsive to user needs.

All three of these require some type of consensus from the community. The problem is to find a facile mechanism.

---

[1]Usually the phrase "research relevancy" implies concern with the relevancy of research to X. Here we are concerned with the relevancy of Gold Coins to research.

MEMO

To: Those Interested in Gold Coins

Date: June 19, 1974

From: McCreight and Simonyi

Location: Palo Alto

Subject: Proposed Golden Utilities and Library Functions

Organization: PARC/CSL

File: UTILITIES

This memo proposes a preliminary set of utility and library functions to be included as Gold Coinlets with Alto operating systems in the future. Standardizing such functions has a number of benefits. It encourages the use of standard representations for data objects. By recycling a single module of code through many applications it represents programming ecology. And since these standard functions are Gold Coinlets, all the wonderful properties of Gold Coins obtain, such as good documentation and maintenance.

A distinction is drawn here between library functions and utilities. Utilities are big monolithic programs which are called infrequently and which communicate with their callers through the file system. Library functions are smaller programs which are called frequently and which communicate with their callers through main memory. This distinction is necessarily and properly fuzzy, and it is intended that there exist a universal library function capable of invoking any utility, and a universal utility capable of calling any library function.

We have tried to estimate the resources needed to implement these utilities and library functions. In many cases an Alto implementation of the function in BCPL or assembly language already exists. In these cases we have assumed that very little new design would be done, and that the existing function would be transliterated into Mesa in a straightforward way. For these conversions our resource estimates should be reasonably accurate, since the difficulty of the original programming is known, and since the conversion is to consist of a simple transliteration. In many other cases no Alto implementation currently exists and considerable design is required in the algorithm and/or the implementation. The resource estimates for these cases are not expected to be very accurate.

UTILITIES

1. Display Editor.

As currently envisioned, the display editor would be able to edit text, edit binary files (leaving semantic interpretations to the user), act as a smart terminal to another computer, act as a calculator (SNOBOL as a subset for editing macros?), and probably even act as EXEC for the Alto Operating System. If Lisp has taught us nothing else, it has taught us that command line interpreters ought to be forgiving and helpful. Thus, although any program which contains a universal utility and a universal library function can be an EXEC, it makes sense for the EXEC to be the smartest interactive program you have. The editor might also include the abilities to print structures in symbolic form and to support a personal file system based on keys and attributes.

2. BCPL System

Until the following facilities are available in Mesa, the following items will have to be available and suported:
a. BCPL Compiler
b. Assembler
c. Loader capable of combining modules produced by (a) and (b)
d. Debugger with some knowledge of BCPL environment

The BCPL compiler, loader, and debugger already exist in final form, except for perhaps one or two man-months of improvements. An assembler has been written in Lisp and another written in BCPL is a few man-weeks from completion.

3. Communication Utility

We will certainly want a utility enabling TTY communication (e. g. TELNET), file transfer (e. g. FTP), age-based automatic file backup, and automatic printing (e. g. XGP part of MINX) among Altos, between Altos and Maxc, and between Altos and the ARPANET. More sophistication would include a SNDMSG facility and smart terminal properties. One might hope, for example, to leave one's Alto in his office and use it from his home terminal via Maxc.

Many of these functions either already exist or will shortly exist in a collection of BCPL programs written by Metcalfe, Boggs, and Rider. EEFTP is a utility which transfers files through the Ethernet. NEWMCA implements a Telnet protocol to Maxc, and supports the Alto end of the MINX file transfer protocol. XPRINT causes a file or set of files to be sent to an XGP-owning Nova for printing. EPRINT sends files to Rider's Alto for printing on Slot.

Metcalfe estimates that transliterating these programs and adding age-based file backup, etc., would take at most five man-months given the availability of a multiprogramming operating system and a Mesa with multiprogramming primitives.

4. Diagnostics

This means a full set of user-runnable hardware diagnostics, for the processor, main memory, display, disk, and Ethernet.

At present only the Ethernet diagnostic and the control RAM loader/diagnostic are written in BCPL; the disk diagnostic and memory diagnostic and display diagnostic are all written in assembly code. Of these only the memory diagnostic really needs to be written in assembly code for precise control of the generated machine code. A new disk diagnostic should be written in a high-level language. However since the current diagnostic is more or less adequate, the rewrite could be in Mesa rather than BCPL. To transliterate the Ethernet diagnostic and control RAM loader, and to rewrite the disk diagnostic ought not to take more than two man-months.

5. A utility for creating bootable files.

One man-week.

6. DUMP/LOAD file bundling

One man-week.

7. Microcode assembler and debugger

This now exists as Nova assembly code. Thacker estimates one man-month to rewrite the assembler part in BCPL. Probably an additional one or two man-months will be required to design and implement a microdebugger suitable for control RAM operation on a stand-alone Alto.

8. Diablo printer utility

Already exists in BCPL; less than one man-week to convert.

9. Librarian program

This program would include facilities to aid in managing the development and maintenance of large software systems. It would include a cross reference generator and structure finder. It would know about versions and compilation dates--and which modules might need to be recompiled if the format of some structure is changed. Some of these facilities, such as cross reference generation, may be more or less independent of the source language in which the software systems are built. Other facilities, such as those which deduce the ramifications of a change in a data structure, would certainly depend on a knowledge of the syntax and semantics of the language.

Deutsch has written a memo entitled "Proposal for a partially automatic source file bookkeeper" describing such a utility. His estimate is that such a program would consist of 30 pages of Lisp code, and would take perhaps two man-months to write.

10. Environment initializer

This program would initialize the state of the Alto and notify the world at large that the Alto is being initialized. For example, it could find out the time, for initializing the Alto's real-time clock. It could broadcast a message of the form "Alto x is being initialized, tell me if you're willing to talk to me" and post the results in a file or table. It could scavenge the disk, and run cursory diagnostics to verify that the machine has no serious flaws. And so on.

This is a very open-ended utility, for which no code now exists. Specifying, designing, and implementing it would probably take on the order of three man-months.

LIBRARY FUNCTIONS AND PROCESSES

1. Environment Inquiries

A whole host of things, ranging from "What time is it?" (real-time standard, probably initialized from Maxc) to "Do I have a Diablo printer?" to "How much display space remains?" to "How many disk I/O references have been made since system startup?" and so on.

This is also very open-ended, should probably be designed and implemented in conjunction with the environment initializer, and would probably take on the order of an additional man-month.

2. Representation Conversion

For example,
a. Number/string conversion
b. Floating-point number/string conversion
c. Time/string conversion

This is somewhat less open-ended, and would probably take less than one man-month after the set of conversions was specified.

3. Storage Suballocator

Less than one man-month.

4. Floating Point

Library routines or microcode are needed for basic floating point operations (+, -, *, /). Furthermore library routines are needed to implement functions like SIN and EXP and ATAN and such like. A tentative decision has been made to adopt as standard representation for floating point numbers the IBM/360 single and double precision formats. This has the advantage of making programs for the various trig functions fairly easy to generate, and it has the disadvantage of losing up to 3 bits of precision in exchange for 2 bits of magnitude. It is also more bulky to implement than a straight binary format, although both formats probably run at comparable speeds.

Leo Guibas has already begun work on this. It is estimated that to implement basic floating point arithmetic with a combination of microcode and Nova machine code will take about two months of Leo's full time. To re-implement the IBM 360 transcendental functions for the Alto ought to take at most one man-month. Leo is not anxious to do this.

5. Streams

The main virtue of streams from the user's point of view is that they conceal the less endearing (and also the more endearing) properties of the I/O devices to which they are connected, making all I/O devices appear the same. Streams should be connectable to disk files or strings in memory or the Ethernet or to keyboard and display. Stream operations include Open and Close, GetItem and PutItem, and Reset to previous position. Stream inquiries include "how big is it?", "where are we now?", and "are we at the end now?". The input and output ends of streams should appear symmetric, particularly if the language system within which the operating system exists supports coroutines (yet another reason why Lisp might want to be written in Mesa).

Somewhat restricted streams already exist in BCPL in McDaniels' operating system. To make them symmetric and to transliterate them into Mesa (probably some redesign will be indicated) ought to take on the order of one to two man-months.

6. Display TTY Simulation

This already exists in assembly code as a part of McDaniels' operating system. However a number of people have indicated a desire for greater flexibility in acquiring display areas and formatting the display. The redesign appears to be the hard part; no matter what is designed, the implementation appears to be straightforward. The implementation ought to take on the order of one man-month after the design is complete.

7. Directory Manipulation

This would do essentially what Tenex does, implementing string to file handle conversions with defaults, version numbers, etc. Facilities available to get next version number, sequence through the files in a directory or a * group, etc. Also able to create and delete files and expunge the disk.

This would probably not require more than one to two man-months to implement, and the design can be pretty much a copy of what Tenex does.

8. I/O Routines

These would build upon the operating system's page-at-a-time, hard disk address I/O routines to implement sequential I/O, direct disk access within partitions, block I/O, and random I/O on sequential files using indexes of pointers. Probably most of the operating system's capabilities for extending files by adding pages, for finding free pages, for marking pages as free, and so on would exist in these routines, rather than in the kernel operating system.

This would probably require two to four man-months, depending on how many features are provided. For example, one feature might be automatically building an index in memory to a file as it is initially read, so that later random reads can proceed without scanning. These routines could be augmented or replaced by those described in Deutsch's memo "A LISP-based file system."

9. Communication Processes

These would build upon the Ethernet's pup-at-a-time I/O to implement stream ends and at least a Telnet and an FTP.

Metcalfe estimates that this will take at most two to three man-months in the presence of a multiprogramming operating system.

MEMO

To:	CSL, SSL	Date: June 24, 1974
From:	Bob Sproull, obedient scribe	Location: Palo Alto
Subject:	Gold Coins	Organization: PARC/CSL

File: GOLD

This tome describes the "Gold Coin" effort about to begin. It is an attempt to outfit Alto's with a beautiful suit of software that anticipates as many users and uses of the Alto as possible.

The chief purpose of this particular exposition is to present the current state of plans, to name the actors, and to solicit reviews. The extraordinary ambition represented by the items mentioned in these pages should not be squandered on projects that are ultimately of little use. Careful and constructive criticism seems the best way to avoid such disappointments.

Don't infer any fierce tyranny in the Gold Coin plan: gold coin software and hardware are intended to be helpful, not authoritative.

The next step in the Gold Coin expedition is to make sure that the plans map well against need and reality. To that end we solicit everyone's input. There will be a meeting Tuesday June 25 at 1:00 to discuss these proposals. Please come.

Definition

A Gold Coin is a dependable, efficient software or hardware facility which services a wide spectrum of Alto-based CSL/SSL computer and systems research activities. Gold coins will be fully supported in the sense that bugs will be repaired, improvements will be made, user-level documentation will be provided, compatibility will be maintained, etc. The set of Gold Coins will cover most of our common needs for tools.

Enumeration

The following is our list of specific facilities that will be included in the Gold Coin set initially:

Alto
Ethernet
Operating System Kernel
Operating System Utilities
Mesa
Interlisp

Since it will take some time to achieve a coherent and complete set of Gold Coins, some interim facilities like BCPL will be supported. It is likely that in the future other facilities will be proposed for inclusion in the set of Gold Coins. The discussion below provides some possible criteria against which these proposals can be evaluated.

Properties of Gold Coins

We have chosen to classify the properties that characterize a Gold Coin under four categories: completeness, research relevancy, longevity, and general usefulness.

Completeness

The set of Gold Coins will constitute the common base of facilities and tools upon which much of CSL's and SSL's research will depend. The set must be made complete enough to support all of our principal research projects by providing those facilities that are used in common. Our initial notions of what constitutes an adequate set of facilities is derived from our present computing environment. The best examples of Gold Coin-like facilities are Tenex and Interlisp. The Alto Gold Coin set must cover our needs at least as well as the Tenex system and subsystems do.

Research Relevancy[1]

Creating and maintaining Gold Coins will require a large commitment of time and energy over an extended period. It is important therefore that Gold Coins for the most part be in the mainstream of our research and not be viewed as extraneous and offensive social obligations. Indeed, as Perlis has argued, one of our principal research objectives should be to learn how to build and maintain systems of Gold Coin quality. But more than that, the principal coins should be congruent with principal research projects in the way that Mesa and Interlisp are. Of course, some Gold Coin-like facilities will not have this intrinsic interest to us, but that need not be a serious problem provided that most of the treasury is filled with gems of research quality. But beware of error 33.

Longevity

If Gold Coins are to provide the bases for ongoing research activities there must be reasonable guarantees that the Gold Coins themselves will be continuously supported for a long time into the future. In addition the Gold Coins must be capable of evolution, both because they or the problems they address are objects of research and because the requirements of the user community will change over time. Thus the keepers of Gold Coins have an obligation to maintain compatibility between new and old versions of their system, or otherwise to help users make a transition between systems.

General Usefulness

We will not attempt to maintain as Gold Coins those facilities that are used solely by individual research projects. Such facilities are the responsibility of those projects. Gold Coin status will be reserved for facilities which are of more general usefulness.

---

[1]Usually the phrase "research relevancy" implies concern with the relevancy of research to X. Here we are concerned with the relevancy of Gold Coins to research.

The fact that Gold Coins will be used widely and that users will not attempt individually to maintain and develop them, imposes some significant responsibility upon the keepers. We should insist on high standards of reliability, repair, and responsiveness to evolving demands. The keepers must furnish documentation at a level appropriate to the average user of the facility, typically at the reference manual level not the primer level. Care must be exercised to insure reasonable compatibility through successive releases and documentation must be kept current. This level of support requires significant resources because the tasks are large and will continue in time. Succession of responsibility must be maintained over the life-time of the facility.

Current Plans for the Projects

This section provides a precis of each of the proposed Gold Coins, together with the names of the individuals likely to be involved, and a time schedule. (This part of the memo is the 'minutes' of the most recent Gold Coin meeting.) More detailed information on several of the projects can be elicited from appendices.

Alto (congratulate Chuck Thacker, Alan Kay, Larry Clark, Mike Overton, Ed McCreight, and others far too numerous to mention)

The Alto Gold Coin is rolling off the production line, and is working very well. They are available by walking up to them. the "Personal Computer" document describing their operation is available from Vicki Parish.

Not all Altos are or will be identical. This is important to bear in mind when considering Gold Coin software: what special requirements, if any, are necessary for a piece of software to run? The two most important considerations are: memory size (48 or 64 K) and the presence or absence of a RAM for microcode.

Virtual Memory (encourage Peter Deutsch, Ben Wegbreit)

A uniform virtual memory addressing scheme should be adopted for the Alto. This notion seems to be endorsed by all proprietors of major languages on the Alto. The current plan is to implement a 24-bit address space: hardware in the memory interface of the Alto will perform the mapping function; if this fails, microcode will look up pages in a hash table in Alto main memory. The hash table records all pages that are present in core. If the requested address is not in core, the microcode gives up, and "paging strategy" code is invoked. This code will vary from language to language. The central notion is that the microcode makes no strategic decisions, it only computes.

Schedule: Approximately 3 weeks are required to complete the functional design, including in particular the interface between the NOVA emulator and the mapping microcode (i.e. NOVA emulator must be undisturbed). An additional 3 weeks are required to design the hardware modifications to the memory interface. A wire-wrap memory board can be constructed in 1.5 months; production versions in 6 months. Software development (hashing, strategy) is complete for LISP now; none will be required for the NOVA emulator. Development will require a RAM and microcode utilities (assembler, debugger).

Ethernet (encourage Bob Metcalfe, David Boggs)

The Ethernet, as a hardware facility, is nearing completion. Hardware interfaces are being constructed at the maximum rate consistent with suppliers and with Gold Coin-style caution. The new transceiver design is nearly finished. A fairly extensive set of diagnostics has already been constructed.

The only significant design activity remaining is to complete the design of PUP (PARC Universal Packet -- see Metcalfe/Boggs memos on this subject). This is an attempt to design data formats and protocols that can be used to transmit among any machines reachable from Alto's, MAXC, etc. (This is a serious endeavor -- please contribute if you have particular needs or prejudices.) The design is expected to be complete by Sept. 1. (See Utilities appendix for description of proposed user software.)

Mesa (encourage Jim Mitchell, et al)

Mesa, the new name for MPS, is advanced as the system-implementation language for Alto. Documentation about the language is available from Jim Mitchell and the Mesa group; an appendix to this document gives a schedule. Generally, the group intends to spend about 10 weeks completing modifications to TENEX Mesa, then to design an interpreter for Mesa programs, then to construct such an interpreter on MAXC and to modify the compiler to produce code for the interpreter. Then, in order to move Mesa to an Alto, the interpreter will be written for the Alto in BCPL or machine language. Thus it is expected that by January, 1975, Mesa will run on an Alto at a speed about 1/20 to 1/50 that of BCPL. Microcoding the interpreter should be done by July 1975. Although this will speed the interpretation considerably, there is debate about the ultimate speed of Mesa interpreted in this way. Estimates range from 1/2 to 1/5 the speed of BCPL, a degradation that some feel is inherent in interpreting a language like Mesa on a machine like the Alto.

Since Mesa is proposed as the Gold Coin system-building language, issues of speed are vitally important to some potential uses of the language. It may simply be that Mesa can never fully supplant BCPL. Those expecting to use Mesa must clearly talk to those expecting to provide it.

The development of Mesa requires specifications for the interface to the operating system kernel, MAXC, and an Alto in October.

<u>BCPL</u> (encourage Howard Sturgis, Dan Swinehart)

Mesa is far enough off that we need to maintain a commitment to BCPL. This is essential to on-going projects in both SSL and CSL.

There is a substantial list of requested modifications to BCPL, ranging from wishes to requirements. Some subset of the modifications will actually be undertaken.

- Language changes:
  - Operations on entities > 16 bits, to facilitate manipulating 24-bit addresses and the like
  - Unsigned arithmetic (esp. compares)
  - Conditional compilation
  - Macros
  - Embedded assignment
  - Modification to switches
- Restrictions or difficulties to be removed:
  - Size of symbol table in compiler
  - Fix loader to handle larger overlays
  - Improve documentation (beautiful document forthcoming)
  - Problems with scope rules for EXTERNALS
- Improvements:
  - Faster compiler, better error recovery
  - Version-checking in source files
  - Loader library features
  - Rewrite runtime system in ASM

The BCPL problem is substantial: John Melvin estimates that 40% of the programming overhead of the POLOS group is 'living around BCPL.' (See Appendix for a more discursive account of BCPL issues.)

<u>Interlisp</u> (encourage Peter Deutsch)

The Alto implementation of INTERLISP is progressing well. Within 3 months, a slow LISP will be available, without the byte interpreter. Within 5 months, a "nice" LISP will be available, that uses byte-compiled functions

for the most part. These will be interpreted by BCPL software with a 20:1 <u>CPU</u> slowdown of byte-compiled code on the Alto as compared to block-compiled code on the PDP-10. A microcode interpreter should change the ratio to about 1:1.

Note: The INTERLISP Gold Coin is intended for use on 64K Altos with Model 44 disks.

The prerequisites for completing the INTERLISP development are: the ability to read the stream position in the operating system, RAM, microcode assembler and debugger, Model 44 disk, a better idea of what display utility routines are desired, and machine-independent sources for INTERLISP (which Warren is graciously providing!).

<u>Operating System Kernel</u> (encourage Butler Lampson, Bill Duvall)

This is the most vague of the Gold Coins; thought, talking and design are still required to specify it. The basic idea is to provide an operating system that: (1) is usable by the Gold Coin languages (Interlisp, BCPL, and eventually Mesa), and (2) meets as many user needs as seems reasonable for any operating system to do. The hope is to design a system that supports multiprogramming, disk access (both as a file system and as a secondary storage device for virtual memory), and which has a library of functions for handling I/O (e.g. Ethernet, disk, Diablo printer, etc.) The system may look like a combination of Noxios and Gene McDaniel's operating system.

When Mesa becomes a coin of the realm, the system will doubtless be recoded in Mesa.

<u>Operating System Utilities</u> (encourage Ed McCreight and all individuals listed in the appendix)

These efforts total about 2 man-years of miscellany. As they all interface tightly with the operating system, the specifications for the system will be needed before the utilities are completed. Please see the appendix for details.

VIRTUAL MEMORY PROPOSAL (Dubbed "alpha")

(At press time, this proposal was undergoing substantial change. New memo by Ben Wegbreit will be available 6/25.)

PROPOSED GOLDEN UTILITIES AND LIBRARY FUNCTIONS

This memo proposes a preliminary set of utility and library functions to be included as Gold Coinlets with Alto operating systems in the future. Standardizing such functions has a number of benefits. It encourages the use of standard representations for data objects. By recycling a single module of code through many applications it represents programming ecology. And since these standard functions are Gold Coinlets, all the wonderful properties of Gold Coins obtain, such as good documentation and maintenance.

A distinction is drawn here between library functions and utilities. Utilities are big monolithic programs which are called infrequently and which communicate with their callers through the file system. Library functions are smaller programs which are called frequently and which communicate with their callers through main memory. This distinction is necessarily and properly fuzzy, and it is intended that there exist a universal library function capable of invoking any utility, and a universal utility capable of calling any library function.

We have tried to estimate the resources needed to implement these utilities and library functions. In many cases an Alto implementation of the function in BCPL or assembly language already exists. In these cases we have assumed that very little new design would be done, and that the existing function would be transliterated into Mesa in a straightforward way. For these conversions our resource estimates should be reasonably accurate, since the difficulty of the original programming is known, and since the conversion is to consist of a simple transliteration. In many other cases no Alto implementation currently exists and considerable design is required in the algorithm and/or the implementation. The resource estimates for these cases are not expected to be very accurate.

UTILITIES

1. Display Editor.

As currently envisioned, the display editor would be able to edit text, edit binary files (leaving semantic interpretations to the user), act as a smart terminal to another computer, act as a calculator (SNOBOL as a subset for editing macros?), and probably even act as EXEC for the Alto Operating System. If Lisp has taught us nothing else, it has taught us that command line interpreters ought to be forgiving and helpful. Thus, although any program which contains a universal utility and a universal library function can be an EXEC, it makes sense for the EXEC to be the smartest interactive program you have. The editor might also include the abilities to print structures in symbolic form and to support a personal file system based on keys and attributes.

2. BCPL System

Until the following facilities are available in Mesa, the following items will have to be available and suported:
a. BCPL Compiler
b. Assembler
c. Loader capable of combining modules produced by (a) and (b)
d. Debugger with some knowledge of BCPL environment

The BCPL compiler, loader, and debugger already exist in final form, except for perhaps one or two man-months of improvements. An assembler has been written in Lisp and another written in BCPL is a few man-weeks from completion.

3. Communication Utility

We will certainly want a utility enabling TTY communication (e. g. TELNET), file transfer (e. g. FTP), age-based automatic file backup, and automatic printing (e. g. XGP part of MINX) among Altos, between Altos and Maxc, and between Altos and the ARPANET. More sophistication would include a SNDMSG facility and smart terminal properties. One might hope, for example, to leave one's Alto in his office and use it from his home terminal via Maxc.

Many of these functions either already exist or will shortly exist in a collection of BCPL programs written by Metcalfe, Boggs, and Rider. EEFTP is a utility which transfers files through the Ethernet. NEWMCA implements a Telnet protocol to Maxc, and supports the Alto end of the MINX file transfer protocol. XPRINT causes a file or set of files to be sent to an XGP-owning Nova for printing. EPRINT sends files to Rider's Alto for printing on Slot.

Metcalfe estimates that transliterating these programs and adding age-based file backup, etc., would take at most five man-months given the availability of a multiprogramming operating system and a Mesa with multiprogramming primitives.

4. Diagnostics

This means a full set of user-runnable hardware diagnostics, for the processor, main memory, display, disk, and Ethernet.

At present only the Ethernet diagnostic and the control RAM loader/diagnostic are written in BCPL; the disk diagnostic and memory diagnostic and display diagnostic are all written in assembly code. Of these only the memory diagnostic really needs to be written in assembly code for precise control of the generated machine code. A new disk diagnostic should be written in a high-level language. However since the current diagnostic is more or less adequate, the rewrite could be in Mesa rather than BCPL. To transliterate the Ethernet diagnostic and control RAM loader, and to rewrite the disk diagnostic ought not to take more than two man-months.

5. A utility for creating bootable files.

One man-week.

6. DUMP/LOAD file bundling

One man-week.

7. Microcode assembler and debugger

This now exists as Nova assembly code. Thacker estimates one man-month to rewrite the assembler part in BCPL. Probably an additional one or two man-months will be required to design and implement a microdebugger suitable for control RAM operation on a stand-alone Alto.

8. Diablo printer utility

Already exists in BCPL; less than one man-week to convert.

9. Librarian program

This program would include facilities to aid in managing the development and maintenance of large software systems. It would include a cross reference generator and structure finder. It would know about versions and compilation dates and which modules might need to be recompiled if the format of some

structure is changed. Some of these facilities, such as cross reference generation, may be more or less independent of the source language in which the software systems are built. Other facilities, such as those which deduce the ramifications of a change in a data structure, would certainly depend on a knowledge of the syntax and semantics of the language.

Deutsch has written a memo entitled "Proposal for a partially automatic source file bookkeeper" describing such a utility. His estimate is that such a program would consist of 30 pages of Lisp code, and would take perhaps two man-months to write.

10. Environment initializer

This program would initialize the state of the Alto and notify the world at large that the Alto is being initialized. For example, it could find out the time, for initializing the Alto's real-time clock. It could broadcast a message of the form "Alto x is being initialized, tell me if you're willing to talk to me" and post the results in a file or table. It could scavenge the disk, and run cursory diagnostics to verify that the machine has no serious flaws. And so on.

This is a very open-ended utility, for which no code now exists. Specifying, designing, and implementing it would probably take on the order of three man-months.

<u>LIBRARY FUNCTIONS AND PROCESSES</u>

1. Environment Inquiries

A whole host of things, ranging from "What time is it?" (real-time standard, probably initialized from Maxc) to "Do I have a Diablo printer?" to "How much display space remains?" to "How many disk I/O references have been made since system startup?" and so on.

This is also very open-ended, should probably be designed and implemented in conjunction with the environment initializer, and would probably take on the order of an additional man-month.

2. Representation Conversion

For example,
a. Number/string conversion
b. Floating-point number/string conversion
c. Time/string conversion

This is somewhat less open-ended, and would probably take less than one man-month after the set of conversions was specified.

3. Storage Suballocator

Less than one man-month.

4. Floating Point

Library routines or microcode are needed for basic floating point operations (+, -, *, /). Furthermore library routines are needed to implement functions like SIN and EXP and ATAN and such like. A tentative decision has been made to adopt as standard representation for floating point numbers the IBM/360 single and double precision formats. This has the advantage of making programs for the various trig functions fairly easy to generate, and it has the disadvantage of losing up to 3 bits of precision in exchange for 2 bits of magnitude. It is also more bulky to implement than a straight binary format, although both formats probably run at comparable speeds.

Leo Guibas has already begun work on this. It is estimated that to implement basic floating point arithmetic with a combination of microcode and Nova machine code will take about two months of Leo's full time. To re-implement the IBM 360 transcendental functions for the Alto ought to take at most one man-month. Leo is not anxious to do this.

5. Streams

The main virtue of streams from the user's point of view is that they conceal the less endearing (and also the more endearing) properties of the I/O devices to which they are connected, making all I/O devices appear the same. Streams should be connectable to disk files or strings in memory or the Ethernet or to keyboard and display. Stream operations include Open and Close, GetItem and PutItem, and Reset to previous position. Stream inquiries include "how big is it?", "where are we now?", and "are we at the end now?". The input and output ends of streams should appear symmetric, particularly if the language system within which the operating system exists supports coroutines (yet another reason why Lisp might want to be written in Mesa).

Somewhat restricted streams already exist in BCPL in McDaniels' operating system. To make them symmetric and to transliterate them into Mesa (probably some redesign will be indicated) ought to take on the order of one to two man-months.

6. Display Utilities

Teletype simulation already exists in assembly code as a part of McDaniels' operating system. However a number of people have indicated a desire for greater flexibility in acquiring display areas and formatting the display. The redesign appears to be the hard part; no matter what is designed, the implementation appears to be straightforward. The implementation ought to take on the order of one man-month after the design is complete.

In addition, there is a need for some form of display routines that can do rudimentary graphics (nothing as complicated as the graphics group's current system). These routines could be used, for example, to "show" data structures as records, lists and the like by drawing boxes, arrows, etc. Again, design is the hard part; implementation is relatively easy. (This might be the "mini-system" that the Graphics Group is contemplating.)

7. Directory Manipulation

This would do essentially what Tenex does, implementing string to file handle conversions with defaults, version numbers, etc. Facilities available to get next version number, sequence through the files in a directory or a * group, etc. Also able to create and delete files and expunge the disk.

This would probably not require more than one to two man-months to implement, and the design can be pretty much a copy of what Tenex does.

8. I/O Routines

These would build upon the operating system's page-at-a-time, hard disk address I/O routines to implement sequential I/O, direct disk access within partitions, block I/O, and random I/O on sequential files using indexes of pointers. Probably most of the operating system's capabilities for extending files by adding pages, for finding free pages, for marking pages as free, and so on would exist in these routines, rather than in the kernel operating system.

This would probably require two to four man-months, depending on how many features are provided. For example, one feature might be automatically building an index in memory to a file as it is initially read, so that later random reads can proceed without scanning. These routines could be augmented or replaced by those described in Deutsch's memo "A LISP-based file system."

9. Communication Processes

These would build upon the Ethernet's pup-at-a-time I/O to implement stream ends and at least a Telnet and an FTP.

Metcalfe estimates that this will take at most two to three man-months in the presence of a multiprogramming operating system.

BCPL ISSUES (by Dan Swinehart)

OCG Needs

The Office Communications Group is devoting most of its efforts to the understanding of the problems of building interactive editing facilities in the NOVA/VTS distributed environment. We have not sensed a great mismatch between the BCPL language and our language needs for accomplishing this task. (This is not a universal opinion; some of us yearn for a very high-level language -- irrelevant for these purposes.) We have, however, encountered serious difficulties in the current compiler and loader configuration:

There are no current software aids for system development by more than one person --

There exist multiple copies of shared sources -- on multiple NOVAS and on MAXC.

It is difficult to assess which sources need compilation even though they have not changed (to accommodate changes to shared declarations ("getted" files), etc.)

There are no current software aids for multi-file development by ONE or more persons --

It is difficult to remember when a file must be recompiled, either because the source has changed or some "getted" source has changed.

No source-comparison and file update mechanism is readily available -- a function of the current NOVA-centered operation, and outside the realm of language considerations, really.

BCPL is quite slow -- one two-page program took 50 seconds, generated 235 (octal) instructions -- just over 3 per second. Lots of that was in the lexical pass, and most of that due to all the "getted" declaration files, most of whose entries are unreferenced in the source.

The compiler often runs out of space, quite often unnecessarily, due to many declarations (again) which don't need to be there (from "getted" files), and more than likely for other curable reasons. When files can simply be broken up, this is usually not too big a problem, but when the number of declarations (legitimate or otherwise) dominates, the problem becomes nearly insuperable.
We currently load our POGOS operating system with our programs, and, lacking any EXEC which we don't provide ourselves, we run that until it breaks, then find the bug, reboot to get DOS back, edit, compile, and reload. The edit/compiler/reload process takes on the order of ten minutes, a large part of it active effort. There are current proposals for improvements to this, requiring the writing of a new BLDR to run as a swapped module in a POGOS image, which could incrementally load or replace user modules (with some restrictions). We have this under control, but some of the proposals below which advocate complete replacement would have to address themselves to this issue.

The debugging environment in POGOS is deficient, but improving rapidly. There are some BCPL changes, mostly the issuance of additional and more complex symbol table information (locals,

structure indicies) which would help, and could be incrementally added.

Remarks on the list above (see page 5):
WSD and others have made specific, medium-to-low priority requests, for changes to the language (e.g., conditional compilation (easy); macros (not too bad, as long as we're conservative about it); switchon changes -- endcase the default, repeatcase and other useful switchon control structures; and multiple-level break/loop facilities, as in SAIL or BLISS. Lack of these features are retarding us no worse than linearly with respect to their predicted value.

CSL Needs

<Howard's list>, the most frightening item on which is the 32-bit quantity, with limited implementation, proposed by LPD. This adds types, not to mention types whose instances are of different sizes, to the language (and, more importantly, to the compiler.)

CSL (LPD and Willie-Sue, at least) also suffer the size restrictions of the current compiler, but not so much the speed ones, or lack of software aids.

Possible Approaches

1. Fight fires, incrementally fixing the compiler to address POLOS needs, joint needs, and perhaps some of those on the CSL list which are easy -- I can envision a 20% devotion of my time to this project with acceptable results. Drawback: Peter doesn't get his 32-bit entities, etc. I favor this approach, based on my predicted commitment.

2. Peter Deutsch rewrites BCPL in INTERLISP, which he can dash out in three weeks (sic), using the tools he already has. Advantages: all new features; great flexibility; potential speed gain sometimes; potential for software aids, esp. wrt time and date storage with output. Drawbacks: Peter's time is more valuable than that; the three weeks sounds a bit optimistic; new programs have bugs; it will probably be too slow when load average exceeds 5.

3. Some sort of joint effort between me and somebody in CSL. Undecidable advantages and drawbacks.

4. I do everything, starting with current compiler -- unacceptable.

5. Mesa group will write sub-Mesa in 6 mo. or so which will run, on NOVAS, better (efficiency, size, debugging) than BCPL. We fight fires in meantime.

6. Charles Simonyi creates a meta-programmer/technician team, OCG designs a new MOL (machine oriented language) by choosing features from a shopping list of systems of that ilk, with emphasis on the debugging aids, software aids, and somewhat incremental compilation/loading/running facilities discussed above; and with 32-bit quantities, etc., for virtual memory on Altos. Would be written to run readily on either machine, etc. We fight fires in the meantime.

Suggestions

I'm hard pressed to make any. In all cases but (2) above, a little bit of (1) will probably have to happen. That's fine. (2) seems quite risky, and an expensive use of Deutsch cycles. (3) would probably not work very well. (5) could easily divert effort from a crucial and already very demanding task: to put full-blown, world-saving Mesa on small machines, probably Altos -- though it might be a forcing function for learning about small machines.(6) sounds promising, since Charles would in any case get what he wants from the effort. It would also provide an interesting testbed for incremental/modular/Mesa-like concepts without the distraction of state of the art control and data structures. It is at least intriguing enough that somebody should talk to C. Simonyi about it. My current main goals are quite short-term -- the next month or two should see changes to facilitate our current editor design.

Many of the software aid goals can be accomplished outside of BCPL/BLDR via offline MAXC programs.

Remarks by Howard Sturgis:

The 32-bit entity request is unreasonable in the light of benefits versus costs (latter extreme).

The "endcase" default is unreasonable because it will either make old programs incompatible with the new BCPL, or require an ugly compiler switch.

Inter-Office Memorandum

To:	CSL/SSL	Date:	June 15, 1974
From:	Ed Satterthwaite	Loc.:	Palo Alto
Subject:	Mesa for the Alto	Org.:	PARC/CSL

On June 12 and 13, the group actively involved in the implementation of Mesa (Geschke, Mitchell, Satterthwaite, Sweet) met to consider the problem of creating a version of Mesa for the Alto. Our conclusions about the major steps as well as time estimates for each are summarized below. An attachment summarizes the more important interdependencies of the steps and indicates a possible division of the work among the members of the group.

We believe that certain additions and changes to TENEX Mesa are essential before Mesa can be moved to the Alto. Most of these are well understood; we propose to implement them in parallel with design of an interpretive system for both MAXC and the Alto.

<u>TENEX Mesa</u>

(1) New version of the segmentation machinery (3 weeks)

(a) complete and test new SEGRUN and associated modules

(b) modify debugger, loader, and bootstrapper

(c) change the compiler to produce modules with new symbol table formats, expanded initialization code

(d) ALSO: extend the compiler to allow arbitrary named types

(2) Finish control structures (4 weeks)

(a) implement support for the control primitives

(b) change compiler's code generators

(c) modify debugger, binder, loader, error handling

(3) General cleanup (3 weeks)

(a) control structures cleanup

(b) implement INCLUDEd program modules, revised binding mechanism

(c) introduction of simple constructors and of sets as data types

(4) Documentation (indefinite)

Alto Mesa

(5) Design the interpretive machine (8 weeks)

(a) instruction set

(b) interpretive engine

(c) Alto microcode feasibility study

(6) Implement interpretive Mesa (i-Mesa) for TENEX (8 weeks)

(a) make an i-Mesa TENEX compiler

(b) make a TENEX i-Mesa interpretive engine

(c) allow i-Mesa and c-Mesa modules to interact

(d) make a complete i-Mesa system for TENEX

Note that this will not be quite identical to an i-Mesa system for Alto (e.g., 36 vs. 16 bit words, different operating system services, etc.)

(7) Move i-Mesa to the Alto (8 weeks)

(a) write a simulator of the Alto operating system interface for TENEX

(b) alter low-level routines of i-Mesa to match the Alto

(c) modify i-Mesa compiler to produce code for the Alto's interpretive engine

(d) modify the TENEX interpretive engine to accept Alto i-Mesa

(e) make a complete i-Mesa system for Alto (running under simulation by TENEX Mesa)

(f) write an interpretive engine in BCPL

(g) transfer i-Mesa system from TENEX to Alto

(8) Move i-Mesa interpreter to Alto microcode (indefinite)

External Constraints

Early in the design of the interpretive machine (step 5) we need to understand the basic facilities to be provided by the kernel operating system for the Alto.

Prior to steps 7a and 7b, we will need a precise definition of that operating system's behavior and interfaces.

Prior to step 7f, we will need one or more dedicated Altos with reliable and reasonably complete (but not highly tuned) utilities and operating system. Easy access to an Alto for familiarizing ourselves with the utilities and service routines would be helpful toward the end of step 5.

Notes

These time estimates, which are thought to be somewhere between realistic and optimistic, imply that a slow but usable version of Mesa could be running on Altos by the end of 1974 (with some luck and few diversions).

They also imply that, with present manpower commitments, further investigation of the substantive issues in the design of Mesa data structuring facilities as well as the implementation of any solutions will be pushed well into 1975.

During the remainder of 1974, we would like to encourage others to begin using TENEX Mesa to the extent that this is possible without a major effort to produce additional documentation.