2 - Highlights of Progress

In the last year, research has progressed on several fundamental issues of AI. As in the past, our research methodology is experimental; we believe it is most fruitful at this stage of AI research to raise questions, examine issues, and test hypotheses in the context of specific problems, such as management of patients with Hodgkin's disease. Thus, within the KSL we build systems that implement our ideas for answering (or shedding some light on) fundamental questions; we experiment with those systems to determine the strengths and limits of the ideas; we redesign and test more; we attempt to generalize the ideas from the domain of implementation to other domains; and we publish details of the experiments. Many of these specific problem domains are medical or biological. In this way we believe the KSL has made substantial contributions to core research problems of interest not just to the AIM community but to AI in general.

Progress is reported below under each of the major topics of our work. Citations are to KSL technical reports listed in the publications section.

2.1 - Knowledge Representation

How can the knowledge necessary for complex problem solving be represented for its most effective use in automatic inference processes? Often, the knowledge obtained from experts is heuristic knowledge, gained from many years of experience. How can this knowledge, with its inherent vagueness and uncertainty, be represented and applied?

Work continues on BB1, with its explicit representation of control knowledge, as reported last year (see the summary of Blackboard Architectures below). In addition, part of our research on NEOMYGIN is focused on using a flexible, rich representation of control knowledge so that we can model problem solving at the strategic level as well as at the tactical level.

[See KSL technical reports KSL-87-01 and KSL-87-32]

2.2 - Blackboard Architectures and Control

How can we design flexible control structures for powerful problem solving programs?

We have continued to develop the BB1 blackboard architecture for systems that reason about -- control, explain, and learn about -- their own actions. In the area of control, we have developed two new domain-independent control capabilities. One generic control knowledge source refines specified parameters of abstract control plans by generating legal values from a semantic network. The other control knowledge source performs opportunistic goal-directed reasoning whenever actions recommended by other control decisions are not executable. In the area of explanation, we have developed the ExAct program. It provides a flexible, menu-driven set of explanation alternatives, as well as a graphical display of the comparative advantages of alternative actions. In the area of learning, we have developed two new capabilities. The WATCH program observes domain experts solving problems and attempts to abstract from their actions the underlying control strategy. It automatically programs new control knowledge sources to generate the hypothesized strategy on subsequent problems. The TRANALOGY program notices when problems in a new domain are analogous to problems in a known domain. It hypothesizes that analogous reasoning methods will work in the new domain as well and automatically programs appropriate knowledge sources.

We have begun conducting various experiments on the costs and benefits of control reasoning. In the context of the PROTEAN system for protein structure modeling, we

are investigating the power of different kinds of control knowledge and strategies to produce computational efficiency. Early results suggest that a small computational investment in control reasoning can produce substantial computational savings in problem-solving operations. We also are exploring differences among alternative architectural realizations of a particular control strategy.

We have continued to develop the ACCORD framework for the class of arrangement problems exemplified by PROTEAN: arrange a set of objects to satisfy constraints. ACCORD substantially enhances BB1's general capabilities for control, explanation, and learning. In addition to PROTEAN, we have applied BB1-ACCORD in the SIGHTPLAN system for designing construction site layouts.

In order to accommodate ACCORD and other task-specific frameworks, we have developed a set of generic framework interpretation procedures for: parsing framework sentences, matching and rating sentences, generating legal parameter values for sentences, and translating sentences into the lower-level language of BB1. These procedures apply to any user-specified framework that satisfies the standards of knowledge and representation laid down in ACCORD. We refer to this growing collection of systems and knowledge modules as the BB* environment.

[See KSL technical reports KSL-86-38, KSL-87-8, and KSL-87-10 and "other outside publications" in Section III.A.3.5]

2.3 - Advanced Architectures

The goals and technical approach of this project, largely supported by DARPA under the Strategic Computing Program, have been discussed in previous annual reports. To summarize briefly, we seek to achieve two to three orders of magnitude speedup in the execution of knowledge-based systems, by identifying and exploiting sources of concurrency at all levels of system design: the application level, the problem solving framework level, the programming language level and the hardware systems architecture level. Due to the inherent complexity of the task and the lack of theoretical foundations for parallel computation with ill-structured problems, we have taken an empirical approach. During the first phase of the project, which will be concluded in July, 1987, we have made specific choices at each of the system levels, i.e. taken a "vertical slice" through the design space, and have conducted several experiments to investigate the effects of a wide variety of parameters on performance.

Some highlights of our accomplishments thus far (most of which occurred during the past year) include:

- Based on a careful and systematic study of potential hardware system architectures, we have established an architectural framework for the underlying machine as a multicomputer array. The study ranged over the full spectrum of possibilities, from shared memory multiprocessors to shared memory multicomputer networks to distributed memory multicomputer networks, taking into account the VLSI opportunities of the 1990's.

- We have designed and constructed a complex, fully instrumented simulator to realize the above architectural framework. The simulated class of machines, called CARE, permits full manipulation of the parameters which specify the hardware system, e.g. communication topology, memory size, etc. CARE is written in Zetalisp, and runs on standard Lisp workstations (TI Explorer, Symbolics 36xx).

- We have studied and implemented basic additions to the Lisp language to accomplish distributed Lisp processing on CARE class machines. These additions are now incorporated into the basic simulation language.

- We created an initial, experimental operating system for CARE class machines, called CAOS. CAOS was used to produce our first experimental results, an end-to-end experiment using the ELINT application, using replicated knowledge sources and pipelining for achieving parallel activity.

- The results of these early experiments were encouraging. Linear speedup, close to the 45 degree line, was achieved up to the intrinsic limits of the application.

- We generalized the traditional blackboard problem solving concept, and developed two new blackboard frameworks. These two frameworks, CAGE and POLIGON, take opposite points of view with respect to the locus of computing activity. CAGE uses knowledge sources as the active agents, whereas POLIGON takes a view that is oriented more towards dataflow, in which the blackboard nodes are the active agents.

- We evaluated a variety of real-world applications as drivers of the underlying system levels, discarding several candidates which initially looked promising but turned out not to be, for various reasons. Consequently, we decided to build our own application, AIRTRAC. As we programmed this application in different problem solving frameworks we began to learn techniques for parallel programming. We initiated experiments to study the performance of AIRTRAC in both blackboard frameworks.

- Detailed studies of the performance achieved in the ELINT/CAOS experiments led to drastic simplification of the pipelining scheme, an orientation toward implementing blackboard nodes as active agents, and using parallel object oriented programming as a low level implementation technique. An environment, called LAMINA, grew out of this analysis. Experiments are in progress to compare the performance of AIRTRAC implemented in LAMINA with AIRTRAC implemented in the blackboard frameworks. The first set of AIRTRAC/LAMINA experiments, using part of the knowledge base that can be used in a data-driven manner, exhibited linear speedup close to the limit of the concurrency inherent in the task.

By the end of 1987 we will have completed five sets of vertical slice experiments. It is already clear that these experiments could have significant impacts on both the hardware and software communities. Specifically:

- One important impact of our research will be to shift the emphasis in parallel architectures for knowledge-based systems from (probably premature) building of hardware to the development of software systems, techniques and tools for the encoding of knowledge-based applications. Hardware can certainly be built. The real difficulty is in developing a firm, quantitative understanding of what hardware actually matters and what hardware may actually hurt (e.g., building hardware based upon incompletely thought-out policy decisions in the software design).

- We will have demonstrated that the distributed memory paradigm is not only a viable alternative to shared memory architectures, but perhaps superior in important ways. The vertical slice experiments provide evidence that implementing a relatively complex application, using a non-shared address space with message passing, can be accomplished without the complexities of managing shared address spaces. Moreover, we will have demonstrated that distributed-memory multicomputers can be programmed to achieve significant (ten to one hundred times) speed-up for nontrivial symbolic problem solving applications. Furthermore, such multicomputer

systems will provide a better fit to the (forecasted) technology for ULSI of the 1990's than the shared memory architectures.

- We will have demonstrated that the major "source of power" in parallel computing is the ability to allow the user to express and manipulate parallel constructs at the level of the application. Thus, the best return on investment is to develop appropriate tools to support parallelism at this level, rather than to support the development of the underlying languages or compilers. The speedup obtainable by only parallelizing programming language constructs in a "programmer transparent" manner (e.g., parallel Prolog or parallel production systems) is very limited.

- An important lesson learned from the success of our simulator is that real applications can be carefully analyzed in an instrumented environment, thereby permitting experimentation with alternate architectures. The community would do well to stress simulation over hardware building.

- We will have demonstrated the need for fast process creation and process switching mechanisms.

[See KSL technical memos KSL-86-36, KSL-86-69, KSL-87-02, KSL-87-07, KSL-87-34, KSL-87-35.]

2.4 - Knowledge Acquisition and Machine Learning

Our research in machine learning has focused on several distinct problem domains including medical (NEOMYCIN/HERACLES) and biochemical (PROTEAN) in addition to domain-independent investigations. We also are motivated by the need for effective tools for knowledge acquisition and maintenance of knowledge bases (IMPULSE and STROBE for FRM, BBEDIT, KSEDIT with BB1).

Several papers by researchers in the KSL were presented at AAAI-86 in Philadelphia in August. Wilkins and Buchanan describe a method of debugging rule sets (see below). Rosenbloom and Laird [14] present a mapping between the SOAR architecture and explanation-based generalization (EBG), in which a justifiable concept definition is acquired from a single training example and an underlying theory of how the example is an instance of the concept. SOAR is an architecture that supports general learning through chunking, which is similar to but not the same as EBG. In addition, the authors suggest answers to some of the outstanding issues in explanation-based generalization.

Chunking is a learning mechanism that acquires rules from goal-based experience. SOAR is a general problem-solving architecture with a rule-based memory that can use the learning capabilities of chunking for the acquisition and use of macro-operators. Rosenbloom et al. are investigating chunking in SOAR and find that chunking obtains extra scope and generality from its intimate connection with the sophisticated problem solver (SOAR) and the memory organization of the production system.

In their AAAI-86 paper, Horvitz, Heckerman, and Langlotz present a framework for comparing alternate formalisms for plausible reasoning [6]. They demonstrate a logical relationship between several intuitive properties for measures of belief and the axioms of probability and discuss its relevance to research on reasoning under uncertainty in artificial intelligence.

Inductive Rule Learning

Buchanan, et al. present an empirical study of the incremental learning process using a careful selection of counter examples in concept formation with the rule-learning system RL (described in last year's SUMEX report). They find that "near misses", negative examples that are similar to acceptable cases, are particularly effective in shrinking the space of possible theories that explain the examples observed. They define a metric for the distance of each example from the target theory and measure the effectiveness and efficiency of examples related to the distance measured, demonstrating that the power of near misses to restrict the space of possible theories results from their small distance from the target. They also find that intelligent selection of instances based upon knowledge of the state of the evolving theory results in a faster convergence of an evolving theory toward the target concept, requiring many fewer cases for learning.

Debugging Knowledge Structures

In large rule-based systems, the performance of the system is strongly dependent on the degree to which the knowledge of the system is "debugged" and refined, i.e., erroneous rules are identified and removed, redundant rules are combined, missing rules are added, and certainty factors of rules are found that give good results over many cases. Such evaluation and restructuring of knowledge is an important type of learning and can be automated to some extent. Here we describe recent work in the debugging and refinement of knowledge bases using several techniques.

Wilkins and Buchanan [19] analyze a problem with the rule sets of rule-based systems that use certainty factors, i.e., better individual rules do not necessarily lead to a better overall set of rules. Since all less-than-certain rules contribute evidence towards erroneous conclusions for some problem instances, the distribution of these erroneous conclusions is not necessarily related to the quality of individual rules. This has important consequences for automatic machine learning of rules, since rule selection is usually based on measures of quality of individual rules. The authors present a method using a new Antidote Algorithm that performs a model-directed search of the rule space to find an improved rule set. They report that the application of this method significantly reduces the number of misdiagnoses when applied to a rule set generated from 104 training instances. This work was also presented at the AAAI-86 Conference in August.

Debugging the knowledge structures of a problem solving agent is the *synthetic agent* method [20] determines a *performance upper bound* for debugging a knowledge base. The synthetic agent systematically explores the space of near miss training instances and expresses the limits of debugging in terms of the knowledge representation and control language constructs of the expert system. This paper presents the framework for evaluating a differential modeling system.

Wilkins describes the ODYSSEUS apprenticeship learning program [21], designed to refine and debug knowledge bases for the HERACLES expert system shell. ODYSSEUS analyzes the behavior of a human specialist using two underlying domain theories, a *strategy theory* for the problem solving method (heuristic classification), and an *inductive theory* based on past problem solving sessions. ODYSSEUS improves the knowledge base for the expert system shell, identifying bugs in the system's knowledge in the process of following the line-of-reasoning of an expert, serving as a knowledge acquisition subsystem. ODYSSEUS can also be used as part of an intelligent tutor, identifying problems in a novice's understanding and serving as student modeler for tutoring systems.

Wilkins, et al. illustrate that an explicit representation of the problem solving method and underlying theories of the problem domain provide a powerful basis for automating learning for expert system shells [22]. By using domain-independent *task procedures* and *task procedure metarules*, domain knowledge can be located and applied to achieve problem solving subgoals. However, these rules are often limited in use due to insufficient domain knowledge. This paper describes the use of *metarule critics* in ODYSSEUS for automating the acquisition of domain knowledge, illustrating a powerful form of failure-driven learning at the level of subgoals as well as at the level of solving the entire problem.

III.A.3.4. Core System Development

1 - Introduction

In this section we describe progress on our core system development and work toward a distributed AIM community. Before launching into the technical details, the motivations and plans for core system work are first summarized along four dimensions: 1) the motivation for the shift of the SUMEX-AIM community from a central mainframe-based model of computing resources to a largely distributed workstation-based model; 2) the prospects for workstation technology and vendor support for a diverse distributed AIM community; 3) the core SUMEX-AIM systems tasks needed to complement vendor developments to realize distributed community operation; and 4) the integration, dissemination, and management of the shift of the AIM community from a centralized to a more distributed operation, including the remaining central resource functions:

- *Motivation for a Distributed Resource:* The motivations for supporting and managing the AIM community as a distributed community are manifest. First the cost/performance trade-offs between centralized shared computing facilities and personal workstations have shifted dramatically toward workstations, especially in the area of interactive symbolic computation resources. While the technology is still quite young, the very best environments for developing knowledge-based systems for biomedicine are arguably already on personal workstations. Various kinds of workstations are rapidly decreasing in cost and increasing in performance so that appropriate models can be selected for cost-effective research support or system dissemination into practical settings like health care clinics or application laboratories.

  Second, the AIM community, with its growing ties into other diverse areas of biomedical informatics, has long been too large to effectively support from a single central node like SUMEX. A number of AIM groups have already moved to local mainframe computing resources (such as at Rutgers University, the University of Pittsburgh, the University of California at Santa Cruz, the University of Minnesota, and Ohio State University). Only some of these have been able to establish network connections for their machines to date, without which low-speed terminal connections must still be made to the central SUMEX resource for mail exchange, software sharing, information access. As workstation prices fall, this trend toward decentralization will accelerate and the need for uniform network access, information services, and systems/software support will increase. The challenge will be to provide responsive central resource services that encourage and facilitate effective communication, collaboration, and information sharing in the new distributed environment.

- *Prospects for Workstation Technology:* Computer workstations have already demonstrated remarkably high performance and low cost for symbolic computing applications. The prospects for future generations of workstations promise an even fuller spectrum of price/performance alternatives. Even with the trend toward more effective personal workstations, however, there are still aspects of an overall computing environment most effectively implemented and supported through central resources. These include services like large-volume information and file storage, special parallel computing architectures, multi-vendor systems expertise, and experimentation with integrating new computing technologies for community deployment. But hardware is only a small part of the

picture -- software represents the larger challenge in the effective integration of workstations with shared resources -- and here is where a community systems integration effort is required. Most vendors are motivated to maximize the sales of their own products, whereas a community of the size and scope of the AIM community must be prepared to integrate technologies from diverse vendors in order to maximize its productivity and to keep abreast of rapidly developing new capabilities. The role of SUMEX-AIM in this new era is to integrate what is available from diverse vendors with core system development efforts to facilitate community research and communications and the smooth evolution of the AIM distributed computing environment.

- *Core Systems Development Tasks:* In order for workstations to support AIM community activities with minimum dependence on expensive, central mainframes, they must be able to supply not only outstanding knowledge-based system development environments but also general computing environments for tasks like electronic communications, text processing, information and file management, and utilities like spreadsheet systems. Many workstation environments do not have fully developed facilities in all these areas and must be augmented. Another major area of core system effort will be in the development of tools to facilitate effective workstation to workstation interactions. These tools include being able to access remote workstation and central computing resources, linking the graphics displays of remote workstations with each other over communication networks, establishing and managing cooperative computing tasks, and enabling remote transfer and sharing of files and information. Finally we must stay abreast of the rapidly changing workstation technology and have allocated a small amount of funding each year to purchase appropriate examples of systems important to AIM community research for testing, evaluation, and development.

- *Managing the Community Transition:* As system research and development progresses, much will remain to be done to integrate and disseminate these new workstation tools throughout the national AIM community -- so that the central DEC 2060 resource can be phased out while maintaining support of community activities. System tools must be tested, evaluated, and refined in the broad context of the AIM community; community groups must fund, acquire, install, and learn to use suitable workstation and network communications equipment; residual central services must be developed and made accessible to support sharing software tools, user consulting, and information resources; and AIM workshop and other management tools for coordinating, integrating, and extending community activities must be evolved. We will use a small group of Stanford and AIM community AI researchers and students to guide the development and testing of distributed subsystems throughout the research period. Initially, these will come mainly from the Stanford community which is easily accessible and has a long experience in experimenting with the development and use of workstation technologies for AI research. After the early years of development and experimental dissemination, we will begin to introduce these tools more extensively for general AIM community use. Our estimate is that these tasks will require the full five-year research period in order to carry out the necessary development, make an orderly and smooth transition, and evaluate the results, without disrupting communications or inter-group collaborations.

2 - Remote Workstation Access, Virtual Graphics, and Windows

2.1 - Remote Access

Lisp workstations of various types have proven extremely powerful, both as development environments for artificial intelligence research and as vehicles for disseminating AI systems into user communities. In addition to the compact, inexpensive computing resources workstations provide, high-quality graphics play a key role in their power. Such graphics systems have become indispensable for understanding the complex data structures involved in developing and debugging large AI systems and are important in facilitating user access to working programs (e.g., for ONCOCIN and PROTEAN). However, as we move towards a distributed workstation computing environment for AI research in the SUMEX-AIM community (and move away from the centralized, shared DEC 2060), a number of technical obstacles must be overcome. One of the most important is to eliminate the need for the user display to be situated close to the workstation computing engine.

This is important in order to allow users to work on workstations over networks from any location -- at work, at home, or across the country. The first step has been getting reliable terminal access operational on all workstations. All workstations now have TCP/IP based terminal servers, and TCP/IP is being installed in the SUMEX network terminal concentrators. This allows primitive (non-graphical) access to the workstation's abilities. A more comprehensive access will be provided through our remote graphics work.

2.2 - Virtual Graphics

In the past, members of the SUMEX-AIM community have often watched each others programs work by linking their CRT terminals to the text output of a running program on the SUMEX 2060. In the case of workstations, though, it is much more difficult to link across several networks to view the complex graphics output of a program. Even locally, it is important to make graphical interaction with workstations across campus or from home possible. One would like to be able to provide the same powerful graphical tools and programming environment that are available to a user sitting in front of the workstation to the remote user if that user has a low-cost bit-mapped display and mouse. In order to accomplish this, it is necessary to capture and encode the many graphics operations involved so that they can be sent over a relatively low-speed network connection with the same interactive facility as if one had the display connected through the dedicated high-speed (30 Mhz) native vendor display/workstation connection.

As reported last year, we studied the feasibility of remote access to workstations by experimenting with a virtual graphics protocol, the Virtual Graphics Terminal Service (VGTS), which was developed at Stanford in the Computer Science distributed systems group [9, 8]. The VGTS provides tools to define objects like windows, lines, rectangles, circles, bitmaps, ellipses, splines, and graphics events like mouse clicks independently of the graphics hardware and operating systems. This encoding minimizes the communication bandwidth required between cooperating hosts, to remotely draw a line, for example.

We also reported that an implementation of this protocol was developed and installed in the operating system of a Xerox 1186 Lisp workstation so that its presence would be transparent to the programmer. This means that if one connects to such a LISP workstation from a SUN workstation (running suitable VGTS software), the Lisp machine graphics will be sent over the net and reconstructed on the SUN workstation without changes to the application program running. This implementation has worked

very well in early experiments so that over an Ethernet, the remote response time is quite close to the response time on the Lisp machine itself.

As a consequence of this work, we had demonstrated the feasibility of remotely using LISP workstations over an Ethernet to take advantage of their graphics programming environment.

During the past year, two new contenders for a virtual graphics standard protocol appeared. These were the MIT Project Athena X window system [15], and Sun Microsystems, Inc.'s Network Extensible Window System [17], referred to as *X* and *NeWS*, respectively. We spent several months studying both X and NeWS and met with representatives of each group supporting these protocols.

X is a very complete protocol that has been developed over the past several years at MIT[1]. X operates at a somewhat lower level than VGP, and as a result can be more bandwidth-intensive. It also assumes a static allocation of computation, display, and interaction responsibilities between server and client. On the other hand, it more fully implements the event mechanisms necessary to track mouse/window interactions and mouse motion histories, and supports color. The protocol has been quite carefully thought out, and provides more flexibility for implementing reasonable emulations of the variety of window systems that exist within our environment. For example, TI Explorers have mouse-sensitive regions within windows called "active regions," and X allows the support for such a region by defining an *Input Only* window with its own cursor. When the mouse moves into such a window, the cursor changes to show the user that he has entered an active region, and at the same time sends an enter-window event to the client. The client can then take the appropriate action for that active region (for instance, scroll text). This is impossible to do in VGP.

NeWS is unique in the sense that it uses a programming language to define its protocol. This programming language is an extension of Adobe's PostScript page layout language for laser printers. This feature gives NeWS its extensibility, for if one wishes to add a new function to the server, one simply sends the PostScript procedure implementing it to the server, and remotely executes that new procedure. This gives the client a great deal more control over what a window looks like; for example, one could implement round or elliptical windows with NeWS. NeWS also allows a client to interact with mouse motion histories and mouse/window events. Thus, it was very difficult to choose between these two protocols.

Ultimately, we chose X as the remote graphics protocol standard for our work. This decision was pragmatic, since we have limited staff resources, and X is receiving wide support from both vendors and the Common Lisp community. An X *client* implementation is being written for Texas Instruments Explorers here at SUMEX-AIM[2]. Our TI Explorer X client is well underway. It is being written in Common Lisp and uses *flavors*, the Explorer object system, to represent instances of X windows. We are currently beta-testing Xerox Common Lisp, and will port the Explorer X client to our Xerox Lisp Machines later this year.

Currently, TI in conjunction with MIT is developing a server implementation for Explorers. DEC is a major supporter of X, and there are implementations under development for their Vax line of equipment. Sun Microsystems is also doing an X

---

[1]The X protocol has been completely redefined this past year. Its most recent version, X.11, is assumed in all of the discussion that follows.

[2]The *client* software runs on the Lisp machine and sends the graphics protocol commands to the remote user display system. The dual of the client is the X *server* software which runs on the user display system and translates the X protocol sent by a client Lisp machine into real graphics pictures and mouse actions.

implementation beneath NeWS, as well as porting X to run directly on their equipment. We are an alpha test site for the SUN implementation. This will provide us with preproduction X server software that we can run on our SUN workstations to aid in debugging our own client software. We anticipate implementations for workstations like MacIntosh II's when a production version of X is released this Fall.

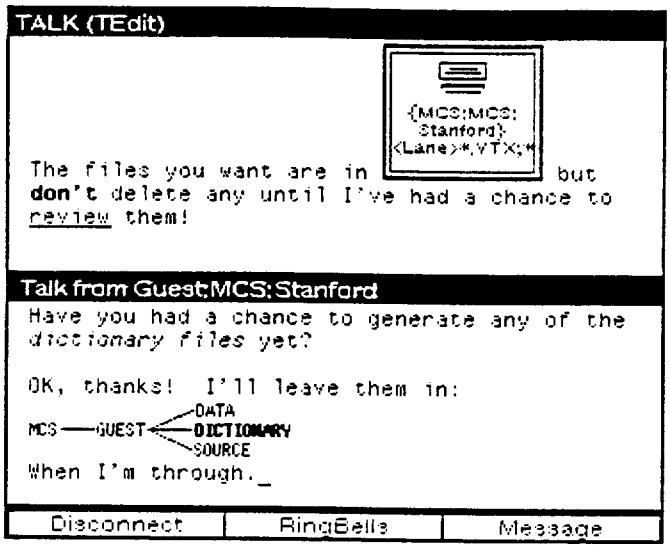
The X window protocol is more bandwidth intensive than some other protocols. It is our feeling that even with this limitation, a suitable subset of the X protocol can be used in cross-country connections where slower communications speeds and longer delays are common. We will have to determine empirically what this subset is. One, for example, would not want to track a mouse in such a situation, but could reasonably expect to use mouse/window events, such as *EnterWindow* or *LeaveWindow*, to manage a remote display over long connection distances. In any case, more work needs to be done in this area to fully develop and integrate these capabilities into Lisp machine systems and to insure that cross-country connections will indeed give usable response time. Success of this work will mean that one can use LISP machine systems from TELENET, ARPANET, or an Ether TIP connection throughout the SUMEX-AIM community.

2.3 - Remote Graphics Applications

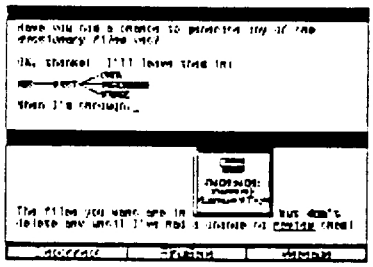
As an example of applying the remote graphics ideas, a TALK program has been implemented which facilitates interactive, electronic communication between users on independent workstations. Layered on the workstation's native editor, the program allows the full use of all editing capabilities in the process of communication, including deletions, corrections and insertions, font changes, underlining, paragraph formatting, etc. Since the workstation's editor also supports both low- and high-level graphics, the program not only facilitates textual exchanges among users, but also allows the sending of screen images (back traces of program breaks, code fragments, etc.) as well as structured graphics images (which can be modified on the destination workstation and returned), all interactively. An example of a TALK session and an illustration of TALK's relationship to other subsystems in the workstation software environment are shown in Figure 2.

The TALK program allows the use of different user interfaces, the workstation's document editor being just one possibility. We also implemented a simpler terminal mode for compatibility with similar programs on other similar and dissimilar workstations. The program was implemented initially using the Xerox XNS family of Ethernet protocols for convenience and speed of development to try out the ideas. Future extensions will include allowing use of different Ethernet (and possibly non-Ethernet) protocols, since the program only requires a reliable byte-stream to operate. We expect the IP/TCP protocols will be added next in order to be able to use the program over the ARPA network.

The TALK program was released gradually to increasing numbers of users in order to get real users' feedback and make changes accordingly. The Medical Computer Science group did an extensive test of the system, where for a period, they used it in place of their normal electronic and non-electronic communication methods whenever possible. This was both a test of the program and an exploration into what people want in the next generation of electronic communication. The TALK program has been released to the Xerox Lisp workstation community as a whole and researchers at Xerox PARC successfully used the program to hold an interactive, graphic, electronic conversation between users at the PARC facility (in California) and Xerox's EuroPARC in England.



Local Host



Remote Host

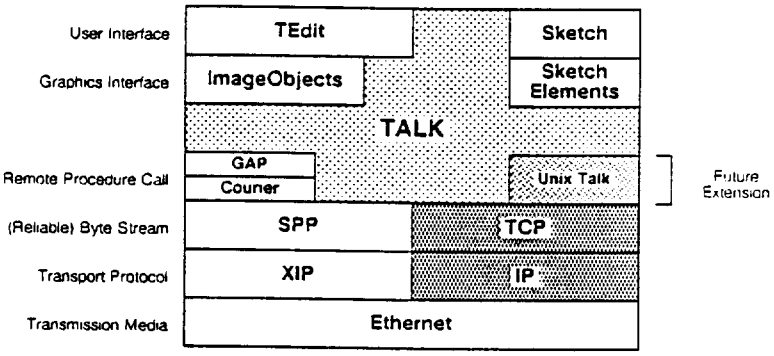


Figure 2: TALK Session Example and the Software Layers Involved in TALK

2.4 - Application-level Window System Standards

Modern programs need to utilize the multiple presentations, non-textual images, and non-keyboard inputs available on all the systems in use by SUMEX. However, up until now, each machine's window system has been idiosyncratic to that machine. There is considerable research now aimed at providing a powerful, flexible window system that can be implemented on a wide variety of hardware, and utilized by many forms of software. However, most of this research is directed at the primitive operations needed to do basic graphics, windowing, and interaction (as in the discussion of X protocols above). We are also working to develop a high level interface to a standard windowing system targeted at the writer of AI applications programs. This system is not being designed to specify the entire man/machine interface, but to provide a simple, easy to understand and useful way for program authors to provide sophisticated interfaces without spending a large percentage of their time working only on the interface. We are currently in the midst of analyzing current applications in order to develop a model for this system based on real-world experience.

3 - File Access and Management

A stable, efficient mechanism for storing and organizing data is central to any computing environment, and is one of the most challenging issues in the move to distributed, workstation-based computing. It is necessary to provide standard services, such as file backup, archival, a flexible, intuitive naming facility, and data interchange services (e.g., software distribution). We also feel that, as the amount of data being manipulated grows, it will become more and more important to have powerful tools for managing hierarchies of files. We plan to support the community with a number of UNIX-based file servers, like the VAX-based servers in use at SUMEX for several years (see Figure 7) and the new SUN-based server (see Figure 5). These will require continued SUMEX-AIM development, however. By keeping the number of servers small, the distributed namespace problem should be manageable in the near term. Current UNIX file servers are relatively cheap and fast. UNIX has many of the needed facilities, e.g., backup, long names, hierarchical directory structure, some file property attributes, data conversion, and limited archival tools. However, while general issues of networking, remote memory paging services, and flexible file access have received considerable attention in both the academic and commercial development of file servers, there seems to be little attention given to other critical operational needs. For instance, the much-used file archiving system of the DEC 2060 (sometimes called off-line cataloged storage) has no analog service in the UNIX systems. Perhaps this is the result of UNIX having its origin in the small computer world where the number of users and volume of data has traditionally been quite low. Our efforts are going into improving the archival facilities and providing case independence and multiple generations by adding SUMEX software between the file system and the network. This should temporarily solve these problems without substantial loss of performance or maintainability.

For the long-term use of the distributed community, we plan to develop an optical disk-based backup and archival system and to use enhanced tools on workstations to do file management. We are currently investigating hardware options for optical disk systems. As better techniques for managing a distributed file system come out of the early research stages, we will use them to improve the distributed file service facilities.

3.1 - Remote File Access

During the past year, there has been a welcomed progress in vendors' attempts to standardize file access protocols. Previously, each vendor had addressed the file storage needs of their particular workstation in a way that was incompatible with most other

workstations, making shared file access and support difficult in a highly heterogeneous environment such as the SUMEX-AIM community. Also, the resources required to maintain many distinct families of filing conventions and protocols on specialized hardware, all meeting the performance needs of a demanding research community, are prohibitive. Thus, last year we proposed to adopt a variant of the *NFILE* file access protocol[1] developed by Symbolics, Inc. It now appears, however, that Sun Microsystems, Inc.'s (SMI) Network File System (NFS) is becoming a more prevalent industry standard, despite the fact that it does not support extensible file attributes and file generations. In order to encourage the porting of NFS to other vendors' workstations, SMI has placed NFS in the public domain, and has a special group dedicated to aiding interested parties in writing the requisite software. This group is also willing to make some changes to the protocol to support non-UNIX file systems (for example, they recently made a change so that NFS could be ported to a CRAY computer). We are now beta-testing a Texas Instruments implementation of NFS on our Explorers, and are ourselves engaged in implementing NFS on Xerox Lisp workstations.

Given that we have acquired an experimental SUN file server this year, and that NFS is supported in the Kernel of the 4.3 release of Berkeley UNIX, this path for unified file access across our mix of workstations appears to be the best solution available. Our anticipated move to 4.3 UNIX on our VAX file servers this summer, and the completion of the NFS port to the Xerox Lisp machines will give us a single file access protocol that is supported by all of our systems with the exception of the Symbolics 3600's. It appears that a third party is working on an NFS implementation for Symbolics machines and we will test this in the coming year.

3.2 - File Server Throughput

At present, a number of file service strategies are employed among and within the various workstation and time-sharing communities. Each strategy has its merits and drawbacks and only in their aggregate do they address all the needs of the users.

One yardstick of utility is the maximum speed of data transfer. Speed of data transfer is affected by the speeds of the processors, disks, I/O circuitry, file system design, network transport protocols, file service protocols, software efficiency, system loading, and other operational parameters. Simple throughput measurements suggest that for the immediate future, the mixed-vendor file service strategy still has advantages from the point of view of data transfer speed. (See Figure 3.)

For the Xerox workstations, the Xerox 8037 file server (using the NS Filing protocol) provided the greatest measured throughput (roughly 37% faster than the Sun 3/180 and Vax 11/750 file servers, using TCP FTP). For the TI workstations, the fastest server was another TI Explorer (using the Chaos FILE protocol) providing throughput 91% greater than the nearest contender (a vax using the Chaos FILE protocol), and 269% faster than the closest IP/TCP contender. The Sun workstation provides a virtual file system interface only for the Sun NFS protocol, and hence was not benchmarked against alternative servers because we are still working on optimized NFS facilities for other workstations and servers.

None of the client/server configurations tested approached the theoretical maximum throughputs projected by disk speeds, network speeds, and other system design considerations. Therefore, we believe that through more effective software engineering it will be possible to simultaneously improve data transfer speed and to reduce the number of server implementations necessary to support the present level of service. For example, the potential for software improvement was illustrated this year by fine-

[1] A *file access protocol* is intermediate between a *remote file system* and a *file transfer protocol.*

tuning of the Xerox implementation of TCP, which yielded improved Sun file server throughput by a factor of 30. In the immediate future, our experiments in this area will focus on the new implementations of the *NFS* file service protocol.

Client	*Server*	*Protocol*	*Reading Throughput*	
DEC 2060	Sun 3/180	TCP FTP	7,000 baud (loaded)	
Sun 3/180	DEC 2060	TCP FTP	17,000 baud (loaded)	
Sun 3/75	Sun 3/180	TCP FTP	55,000 baud (unloaded)	
Xerox 1186	DEC 2060	PUP Leaf	18,181 baud (loaded)	
Xerox 1186	DEC 780 (VMS)	TCP FTP	33,402 baud (loaded)	
Xerox 1186	Xerox IFS	PUP Leaf	52,526 baud (unloaded)	
Xerox 1186	DEC 750 (UNIX)	PUP Leaf	53,036 baud (loaded)	
Xerox 1186	DEC 2060	PUP FTP	67,001 baud (loaded)	
Xerox 1186	Sun 3/180	TCP FTP	71,192 baud (unloaded)	(was 2,412 baud)
Xerox 1186	DEC 2060	TCP FTP	72,207 baud (loaded)	(was 2,850 baud)
Xerox 1186	DEC 750 (UNIX)	TCP FTP	72,412 baud (loaded)	(was 9,096 baud)
Xerox 1186	Xerox IFS	PUP FTP	84,125 baud (unloaded)	
Xerox 1186	Xerox 8037	NS Filing	103,519 baud (loaded)	
Xerox 1186	Xerox 8033	NS Filing	105,486 baud (unloaded)	
Xerox 1132	DEC 2060	TCP FTP	3,228 baud (loaded)	
Xerox 1132	DEC 2060	PUP Leaf	18,737 baud (loaded)	
Xerox 1132	DEC 750 (UNIX)	PUP FTP	75,361 baud (loaded)	
Xerox 1132	DEC 750 (UNIX)	PUP Leaf	81,711 baud (loaded)	
Xerox 1132	DEC 750 (UNIX)	TCP FTP	121,163 baud (loaded)	
Xerox 1132	DEC 2060	PUP FTP	167,687 baud (loaded)	
Xerox 1132	Sun 3/180	TCP FTP	215,000 baud (loaded)	
Xerox 1132	Xerox 8037	NS Filing	234,154 baud (loaded)	

			Reading Throughput	*Writing Throughput*
TI Explorer	DEC 750 (UNIX)	TCP FTP	36,952 baud	96,000 baud
TI Explorer	Sun 3/180	TCP FTP	58,888 baud	135,208 baud
TI Explorer	TI Explorer	TCP FTP	61,376 baud	121,512 baud
TI Explorer	DEC 2060	TCP FTP	63,320 baud	110,592 baud
TI Explorer	DEC 750 (UNIX)	Chaos FILE	122,136 baud	129,376 baud
TI Explorer	TI Explorer	Chaos FILE	233,008 baud	221,192 baud

Figure 3: File Server Throughput Benchmarks

4 - Electronic Mail

Electronic mail has become a primary means of communication for the widely spread SUMEX-AIM community. The advent of distributed workstations is forcing a significant rethinking of the mechanisms employed to manage such mail. With mainframes, each user tends to receive and process mail at the computer he uses most of the time, his *primary host.* The first inclination of many users when an independent workstation is placed in front of them is to begin receiving mail at the workstation, and, in fact, many vendors have implemented facilities to do this. However, this approach has several disadvantages:

- Workstations (especially Lisp workstations) have a software design that gives full control of all aspects of the system to the user at the console. As a result, background tasks, like receiving mail, could well be kept from running for long periods of time either because the user is asking to use all of the machine's resources, or because, in the course of working, the user has (perhaps accidentally) manipulated the environment in such a way as to prevent mail reception. This could lead to repeated failed delivery attempts by outside agents.

- The hardware failure of a single workstation could keep its user "off the air" for a considerable time, since repair of individual workstation units might be delayed. Given the growing number of workstations spread throughout office environments, quick repair would not be assured, whereas a centralized mainframe is generally repaired very soon after failure.

- It is more difficult to keep track of mailing addresses when each person is associated with a distinct machine. Consider the difficulty in keeping track of a large number of postal addresses or phone numbers, particularly if there was no single address or phone number for an organization though which you could reach any person in that organization. Traditionally, electronic mail on the ARPANET involved remembering a name and one of several "hosts" (machines) whose name reflected the organization in which the individual worked. This was suitable at a time when most organizations had only one central "host." It is less satisfactory today unless the concept of a "host" is changed to refer to an organizational entity and not a particular machine.

- It is very difficult to keep a multitude of heterogeneous workstations working properly with complex mailing protocols, making it difficult to move forward as progress is made in electronic communication and as new standards emerge. Each system has to worry about receiving incoming mail, routing and delivering outgoing mail, formatting, storing, and providing for the stability of mailboxes over a variety of possible filing and mailing protocols.

Thus, we are investigating the alternative strategy of having a *mail server* machine which handles *mail transactions.* Because this machine would be isolated from direct user manipulation, it could achieve high software reliability easily, and, as a shared resource, it could achieve high hardware reliability, perhaps through redundancy. The mail server could be used from arbitrary locations, allowing users to read mail across campus, town, or country using more and more commonly available workstations.

The mail server acts as an interface among *users, data storage,* and *other mailers.* Users employ a *mail access protocol* (MAP) to retrieve messages, access and change properties of messages, manage mailboxes, and send mail. This protocol should be simple enough to implement on relatively uncomplicated, inexpensive machines so that

mail can be easily read remotely. This is distinct from some previous approaches since the mail access protocol is used for all message manipulations, isolating the user from all knowledge of how the data storage is used. This means the the mail server can utilize the data storage in whatever way is most efficient to organize the mail. The data storage could be anything from conventional magnetic disk file system to a highly specialized mail filing system built on optical disks, since it is abstracted from other elements in the mail system. The other mailers constitute the mail server's (and thus the users') link to the outside world. The mail server would use various *mail transport protocols* (e.g., SMTP) to exchange mail with other mail hosts.

We have been investigating user/mail interface issues for workstations, as well as issues for the mail access protocol itself. We are examining several related projects, including MIT's PCMAIL (Mark Lambert, MIT Distributed Systems Group), the public parts of Xerox's Grapevine and NSMail, and work on Stanford's V system.

We have implemented an Interim Mail Access Protocol (IMAP) server on the 2060 and a client implementation in Interlisp on Xerox D-machines. The resulting beta-test mail environment proved to be quite usable; some D-machine users use it as an alternative to the 2060 mail environment in their daily mail work.

The IMAP server manipulates the actual file store copy of the user's incoming electronic mail under direction from the IMAP client. As noted above, the client has no knowledge of the (possibly operating system- dependent) format of mail on the server's file store; the IMAP protocol provides its own representation of mail and the server translates between this and its host system file store conventions.

The IMAP client issues a series of *fetch* commands to retrieve data from the server. A fetch command has two arguments: a *message sequence* and the name of the data item to be fetched. A message sequence can be a single message number, a range of message numbers, or a list of numbers or ranges. For example, a typical fetch command might be "fetch 2:7,10 flags", meaning "fetch the status flags for messages 2 through 7 and message 10" (status flags include "new message", "deleted message", "message has been read", etc. as well as user-defined flags).

In IMAP, the actual message data is identified by names such as "RFC822.Header" and "RFC822.Body" referring to the text-based mail representation used on the DoD Internet standard (RFC 822). This is intended to be a temporary solution only, since RFC 822 lacks structure and the capability to deal with non-text mail. We plan on extensions to IMAP (IMAP II, see below) that will introduce a canonical and structured representation of an electronic mail message. In such a structured form, an electronic mail message would consist of a set of named properties and property values.

During implementation of the user interface we observed that the IMAP protocol had several deficiencies which made certain mail concepts difficult or impossible to implement. For example, there is no way in IMAP to notify the client of newly arrived mail during an IMAP session. Other IMAP deficiencies were observed in the design of a Common Lisp implementation for Texas Instruments Explorers; in particular, IMAP is a "lock-step" protocol with no mechanism for multiplexed operation. This means that IMAP is vulnerable to synchronization problems in which a client interprets part of a previous response as the answer to the current query.

To address these concerns, a new Interim Mail Access Protocol (IMAP II) was designed after extensive review. IMAP II is heavily influenced by IMAP, although with a greater degree of formality in the specification and quite a bit more extensibility. Instead of the lock-step query/response model of IMAP, IMAP II uses tagged commands and data and explicitly allows unsolicited data to be sent from the server to the client. IMAP II introduces a more formal structure to server-to-client path; all data is now identified unambiguously. This is especially important for extensibility and unsolicited data.

In addition, IMAP II makes it possible to fetch more than one item of data at a time. This is an important performance issue since often the client needs to fetch a set of data items for a set of messages. The IMAP model of fetching a single data item at a time resulted in the client having to make several consecutive requests with much longer turnaround than a single request that specifies everything the client wants.

A large subset of IMAP II has been implemented on the 2060 by modifying the existing IMAP implementation. Both the 2060 implementation and the specification have been left open-ended to allow for extensions as the need arises. Work is now in progress to modify the Interlisp user interface to use IMAP II. Since the interface is no longer limited to the model of IMAP a general restructuring of the Interlisp client is being done to take advantage of the new facilities offered by IMAP II. A Common Lisp implementation, based on IMAP II, is also in progress.

5 - Text Editing

All workstation systems have text editing facilities, some adaptations of systems in use on mainframes (e.g., EMACS-like editors) and some specialized What-You-See-Is-What-You-Get (WYSIWYG) editors (e.g., TEdit for Xerox workstations or InterLeaf for UNIX workstations). We are currently making use of each workstation's facilities, making extensions where needed to bring compatibility among the various workstations (in both user interface and document format) without detracting from the powerful, but idiosyncratic, features. Text formatting, to produce printable or displayable forms of documents, is another area where considerable vendor effort is expended. Implementations of SCRIBE or TEX systems are available for some workstations directly. Also, since these formatting processes are essentially batch operations, we expect to provide servers that offer formatting services. These can be fed a raw manuscript and will return a formatted version, suitable for one of the several printing device standards in use. WYSIWYG editors are able to combine the editing and formatting processes into the document preparation system. We will concentrate on PostScript and ImPress printers, allowing Press printers to fade from use. The 2060 also provides for printer spooling, based on a first-come-first-serve algorithm with priorities determined by submission time and estimated pages of output. This spooling is not available among workstations currently. Given adequate printing resources, a laissez-faire access policy without spooling can work adequately. If there is a problem, an arbitration scheme will need to be worked out, but this should be a relatively straightforward task. Finally, we will need to augment vendor products to provide essential text processing aids for functions like spelling correction, document merging and segmenting, and document analysis.

5.1 - Text Processing for Xerox D-machines

TEdit is the text editing and formatting package on the Xerox D-machines (i.e., the 11xx series) and we have continued our work to extend this environment to displace text processing from the DEC 2060. Almost all efforts during the past year were directed towards the Interlisp package TMAX. TMAX stands for Tedit Macros And eXtensions and it gives TEdit the ability to do things that hitherto could only be done with Scribe. Scribe is a powerful document preparation language, but it consumes far too many mainframe cycles. Furthermore, with Scribe you must hardcopy your output to see what it looks like. TEdit is a WYSIWYG text editing and formatting system, which means that you can see what your output will look like while you are creating it.

TMAX makes no attempt to mimic Scribe in TEdit. This would be a Herculean task given the power and flexibility of Scribe. Instead TMAX implements some of the more commonly used features of Scribe, including indexing, numbering, end notes, and forward/backward referencing. All of these features are implemented through menus.

For example, to include an index request in a document, the user simply buttons the Index command (with the mouse) and then types the text to be indexed. TMAX takes care of all the rest (e.g., associating the page number with the indexed text, creating a sorted list of the indices, etc.). These TMAX features plus the editing and formatting features already available in TEdit make the TMAX/TEdit package an attractive alternative to Scribe.

The following is a quick overview of the major TMAX features:

- Indexing -- users can insert both simple and extended index requests, create a sorted file of the indices and their page numbers, and even specify that the page numbers be printed in manual format (e.g. III:25.7 for chapter 3, section 25, page 7). A simple index is just the text to index. An extended index takes the text to sort on, the text to print, the font to print it in, and a page number option. This option allows the user to specify the normal page number in the index file, no page number, or a user specified fixed page number. There is also a command that pops up a menu of the simple and extended indices specified so far and users can insert additional index requests by simply buttoning the corresponding item in this menu.

- Numbering -- users specify the names and order of "number markers" and then insert these markers wherever they want something numbered. Users can create as many different number markers as they like and some can be layered (i.e. chapter, section, etc.) while others are disjoint. When a marker is inserted or deleted, TMAX automatically adjusts all the related numbers. Users can even specify the font and format of each number. The format defines how the number will be displayed (i.e. an Arabic or Roman numeral or a letter), the delimiter following the number, and the starting value. There is also a facility to create a standard table-of-contents file.

- End Notes -- these are just like footnotes except end notes are inserted at the end of the text rather than the bottom of the page. A future version of TMAX will support footnotes. When an end note is inserted or deleted, TMAX automatically adjusts the other end note numbers.

- References -- users can refer to specific numbering markers or end note numbers by their numeric value. It does not matter if the number is before or after the reference to it. Also, should a number change because a number marker or end note was deleted or inserted, the reference to that number will be automatically updated (as well as the number itself).

There are many more features and options in TMAX and still more in the planning stages. For example, one can edit the text of an end note by pointing the mouse to the end note number and pressing the middle button. Another TEdit window will appear containing the end note text. Some of the features planned are footnotes, bibliographies, and appendices. The TMAX User's Guide describes all the features of this package.

5.2 - Remote Editing

Currently, the mainframe editor of choice among our users is EMACS. EMACS, like Scribe, is very powerful but it also places a heavy load on our mainframe. In an effort to reduce the mainframe load (and ease users into using TEdit), we have written WEDIT (Workstation EDITor). WEDIT provides a convenient way for mainframe EMACS users to edit their files on a Xerox D-machine using TEdit. Note that WEDIT itself is not an editor. It simply opens a connection to the workstation and sends a

packet containing the name of the file to edit. The workstation does all the rest. When the user is done editing, the workstation sends the updated file back to the mainframe. From the mainframe's point of view WEDIT is an editor in the sense that given a file, it returns an updated version. Because of this, it is easy to install as the default mainframe editor. A simple change in the user's login command file is all that is necessary. From that point on, each time the user edits a file, the editing will be done by TEdit on the user's personal workstation. EMACS users can experience long delays when the 2060 is heavily loaded. With WEDIT (i.e. TEdit) there are no delays since the editing is done on the user's personal workstation.

5.3 - Special Document Types

In last year's report, some TEdit extensions to facilitate simple document types like memos were mentioned. These extensions proved to be very useful although this package was only a prototype. Using the concepts developed in this package, we have written a new TEdit package called *Letterhead*. The Letterhead package allows users to create standard letterheads, for example, for Stanford University correspondence. All the options in this Letterhead package are menu driven. When a user starts the Letterhead package, a TEdit window appears on his workstation and the user is prompted for several different fields. First a menu of the possible Stanford logos pops up and the user must select one of these logos. The logo is placed in the upper left hand corner of the window. Next a menu of the return addresses pops up. The user may select one of the known addresses or create his own. Next the Letterhead package asks the user how the address should be justified. This is done though a menu and the possible ways are left, right, or centered. The justified address is then inserted in the upper right hand corner of the window. Finally, the current date is automatically inserted just below the logo. Now the letterhead is complete and TEdit is ready to accept input from the user. The user can change either the logo, address, or date by pointing the mouse at the appropriate field and pressing the middle button. If the logo is buttoned, the logo menu will pop up and the user can select a different logo. If the address is buttoned, the return address menu will pop up and the user can either select a known address, create his own address, or edit the address already in the document. If the date is buttoned, a date menu pops up allowing the user to display the date in one of several different formats. When this window is hardcopied, it will look just like a standard Stanford University letter

6 - System Building Tools

Traditionally, a large set of languages and programming environments have been supported on the 2060 in order to encourage experimentation and development. We now believe that the experience gained in those years of broad experimentation can be distilled into a fairly small set of languages and tools, relieving the researcher of the need to learn many programming languages, while still providing the needed facilities to allow the experimentation to move further into the higher levels of knowledge representation systems and problem solving architectures. As we move to the workstation based environment, we plan to phase out support for many of the languages we have offered in the past and concentrate on the most relevant languages for AI research and applications: C, FORTRAN, InterLisp-D, ZetaLisp, and Common Lisp. Common Lisp has already achieved popularity as a standard (see page 54), and many projects are already using it. We expect to press for further adoption of Common Lisp as a community symbolic computing standard, consistent with prior investments in large software systems such as those which exist for on-going AIM projects. In addition, we will support important higher-level knowledge representation and problem solving architectures (e.g., S.1, KEE, Strobe, and others) as appropriate for community research and dissemination activities.

7 - Distributed Information Resources and Access

There are many user needs for getting information from and about the computing environment, ranging from help with command syntax to sophisticated database queries. A distributed computing environment adds new complexities in making such information accessible and also new requirements for information about the distributed environment itself. We are adapting the many workstation-specific information tools to include distributed environment information such as workstation and server availability, "finger" information about user locations and system loads, network connectivity, and other information of interest to users in designing approaches to carrying our their research tasks. In addition, we will have to develop general systems tools for monitoring and debugging distributed system performance to identify workstation and network problems. Finally, we must adapt and develop distributed system tools for remote database queries and organize the diverse sources of information of interest to AIM community members to facilitate remote workstation access to community, project, and personal information that has traditionally existed in ad hoc files on mainframe systems.

In conjunction with the SUN file server we have been integrating, we have mounted an experimental database system for remote information access using the commercial UNIFY database product. Our goal is to make access to the database information possible from a distributed workstation environment through network query transactions, as opposed to asking the user to log into the database system as a separate job and type in queries directly. This will facilitate remote information access from within *programs*, including expert systems, where the information can be filtered, integrated with other information, and presented to the user. The system will provide multi-user, multi-database access capability; that is, several users will be able to have access to a single database at the same time, and a single user will be able to have access to several databases at the same time.

The initial implementation of the remote query system was done on a TI Explorer. The query interface on the Explorer communicates with the Sun UNIFY database system via the Remote Procedure Call (RPC) mechanism which underlays the NFS remote file access system. The Explorer calls a server on the SUN and sends an SQL/DML query command as an argument to a remote query procedure, and receives the retrieved data and/or a message sent back from the server. SUN UNIFY can already manage multiple databases, so a client can have several databases open at the same time. The operations on the database are transaction-oriented, and therefore the concept of a database access session is applicable. The access functions currently implemented are *open a database, close a database, retrieve data from a database, insert records into a database, delete records from a database, update the database, lock a database,* and *unlock a database.*

This facility can be easily converted to run in Lisp environments on machines with SUN RPC services implemented. Currently, there is no RPC package for the Xerox D-machines, so we undertook implementing one. This should be done by early summer.

8 - Distributed system operation and management

The primary requirements in this area are user accounting (including authorization and billing), data backup, resource allocations (including disk space, console time, printing access, CPU time, etc.), and maintenance of community data bases about users and projects. Our accounting needs are a function of NIH reporting and cost recovery requirements. The distributed environment presents additional problems for tracking resource usage and will require developing protocols for recording various kinds of usage in central data base logs and programs for analyzing and extracting appropriate reports and billing information. We are now involved in analyzing the kinds of

resource usage that can be reasonably accounted for in a distributed environment (e.g., printing, file storage, network usage, console time, processor usage, server access), and investigating what facilities vendors have provided for keeping such accounts. Data backup is, of course, closely related to the filing issue. We continue to use and improve network based file backup for many of our file servers.

9 - Mainframe and Workstation System Environments

The various parts of the SUMEX-AIM computing environment require development and support of the operating systems that provide the interface between user software and the raw computing capacity. This includes the mainframe systems and the workstation systems. Following are some highlights of recent system software environment developments.

9.1 - TOPS-20

Our long-term plan to phase out the 2060 mainframe system has continued as scheduled. Development efforts on the 2060 have ceased, except where needed to keep the machine operational in the evolving distributed environment. This involves considerable work in areas such as file system archiving, retrieval, and backups; periodic updating, checkout, and installation of new versions of system software; the regular maintenance and updating of system host and network tables; and monitoring of and recovery from system failures, both hardware and software. Over the past year, the main areas of activity include:

- *Network service reliability* -- The SUMEX 2060 has experienced relatively frequent software crashes resulting from system problems in the handling of free space by the IP/TCP network software. During periods of heavy use, the entire system would suffer an unscheduled restart approximately every eighteen hours. After a considerable amount of investigation of crash dumps, we isolated a cause. The problem was introduced over a year before in a modification made by another site in an attempt to improve network performance. After fixing this illusive bug, the 2060 reliability has improved markedly and the system regularly runs for over a week between reloads.

- *Network naming domains* -- The Internet community is in the process of converting to a domain naming scheme, to replace the flat address space of the old exhaustive host tables prepared by the Network Information Center. Although we have converted to using only fully qualified names, we are not yet running the domain system on the 2060. This is due in part to the unreliability and incompleteness of the domain software for TOPS-20's at this point. We expect to move to full domain support this coming year.

- *Dial-up communications* -- A significant portion of work on the 2060 is carried on via dialup modems from homes. During the past year we rearranged and consolidated our incoming modem lines. We combined several inside and outside phone number hunting sequences serving several different modem types and speeds, into well defined groups for old-style Vadic 1200 modems, local versions of split speed modems, and other types. This last group serves any Bell/CCITT modem at any speed from 300 baud to 2400 baud. During this process we removed all the outside phone lines, and now operate exclusively through Stanford-operated SL100 lines. In addition to these mainframe modems, we have installed 10 modems on an Ethernet TIP, allowing users, once dialed in, to connect to the host of their choice.

- *Cost Center accounting* -- During the past year, the 2060 accounting programs were updated to reflect the new Cost Center structure (see Section III.D.2). All the various users and projects were organized according to their cost center account numbers, and monthly reports are generated to reflect this usage. As part of this conversion, a concerted effort was made to review all of the SUMEX accounts, and remove those that were otherwise no longer appropriate.

9.2 - UNIX

We run UNIX on our shared VAX 11/750 file servers. This system has been used pretty much as distributed by the University of California at Berkeley, except for local network support modifications, such as for ChaosNet protocols. The local VAX user community is small, so we have not expended much system effort beyond staying current with operating system releases and with useful UNIX community developments.

9.3 - Xerox D-Machines

Much of the SUMEX-AIM community continues to use InterLisp, including many Dandelion (1108), Dandetiger (1109), and DayBreak (1186) machines, in addition to the Dorado (1132). We have used the Xerox implementation of the TCP network protocol (in cooperation with Xerox) extensively this past year and saw its performance and reliability improve a great deal. We began a Lisp implementation of Sun NFS (*Network File System*). The ARPA protocol suite, which is seeing increasing usage, lacks a mechanism for random file access or attribute manipulation. The Sun specification partially fills this void and appears to be a standard whose acceptance is growing.

The Interlisp software remained stable this year and almost no user time was wasted on software revision problems. A number of new utilities were written locally or acquired from other sites with whom we exchange expertise on the ARPA Internet.

We are among the first users of Xerox Common Lisp for the Xerox Lisp machines. The advantages to our community are early availability of this widely-recognized dialect of the Lisp language and the ability to specially direct the implementers' attention to the problems of greatest concern to us.

The Info-1100 discussion list which we sponsor saw another year of growth of readership and participation on the ARPA Internet, Usenet, Bitnet, and CSNet. Among the beneficiaries are other NIH-sponsored projects at Ohio State University and the University of Maryland.

In conjunction with the Info-1100 mailing list, a library of user-written software is made available to the Internet community on the SUMEX-AIM 2060 computer. Over 60 packages and supplements were distributed this way. Additionally, the source code to many of these packages was mailed to the Info-1100 mailing list in order to reach an even wider group.

We have worked closely with many other sites, including the Center for Study of Language and Information at Stanford, the Stanford Campus Networking group, Rutgers University, Ohio State University, the University of Pittsburgh, Cornell, Maryland, and industrial research groups such as Xerox Palo Alto Research Center, SRI, Teknowledge, IntelliCorp, and Schlumberger-Doll Research. We have been the maintainers for the international electronic mail network of users for research D-machines, which have upwards of 300 readers, and the interchange of ideas and problems among this group has been of great service to all users.

Although numerous Xerox Lisp machine sites are able to obtain software from SUMEX-AIM via anonymous FTP over the ARPA network, it became increasingly clear that a large part of the community did not have such access even though there is electronic mail connectivity. To experiment with distributing software to these sites, we put together a simple ASCII encoder for binary files, BMENCODE. This program makes it possible to mail binary files (TEdit editor files and *COM files from the compiler) to isolated sites, exploiting Interlisp's inherent ability to encode bitmaps into ASCII files. Numerous files were successfully transferred around using this program. As the user community has begun to see the value of such a utility, more efficient versions of the program have been developed elsewhere.

In extending our XNS boot service (which provides installation and diagnostic programs for our workstations) to work with the new 1186 Hardware, we ran into trouble as the 1186's hastily written initial Ethernet microcode. The booting sequence violated Ethernet layering principles which prevented it from routing beyond the local network. After nearly a year of exchanging letters, packet traces and software with Xerox, the problem is still unresolved. This led to our adding a second Xerox 8000-based XNS boot server (using a spare 1108 processor) to our other major network with 1186 hardware. This additional server provided a suitable work-around to the problem and only a single workstation is still unable to access network boot services.

Our move to a new building this past year involved the de-installation and reinstallation of nearly thirty workstations plus several printers and other servers. In anticipation of the move, diagnostics were run on all of the Xerox University Grant 1108s in order to get any existing problems fixed under warranty. The diagnostics were run again after the machines were installed in the new facility. All the equipment was successfully relocated without major incident.

9.4 - Texas Instruments Explorers

The twenty Texas Instruments Explorers have enjoyed an increasing popularity as more projects have developed a need for the combination of execution speed, full Common Lisp, and sophisticated development facilities offered by the Explorer. Explorers have come into use in other parts of the national biomedical community as well, such as Ohio State University and the University of Maryland. However, the Explorer is still maturing as an AI workstation. Thus, our efforts have been directed at improving the environment of the Explorer by developing software, organizing user interest activities, and advising Texas Instruments.

Previous experience has shown that the greatest source of advancement for a particular computing environment is the user community. They are the most in touch with the deficiencies of the system, and thus uniquely positioned to address them, as well as to utilize the strengths of the system. The product developers of the system are frequently too involved in the lower levels of detail to produce general, effective solutions to problems, as well as being hampered by limited manpower resources. However, a significant amount of time and effort is required to organize this effort. This task has traditionally fallen to a user-run organization, such as DECUS or Usenix.

We are spearheading the effort to organize a national or international users' group for the Explorer. The goals of this undertaking are to:

- facilitate dissemination of information by organizing meetings where presentations and discussions can be used to make little-known techniques and facilities more widely known, as well as feeding back information on needs and wants to developers,
- allow more immediate communication via electronic mailing lists, which are

used for distribution of important software fixes and discussion of items of general interest, such as new software tools, or proposed changes to the system,

- publish a periodic newsletter containing usage tips, salient extracts from the electronic mailing lists, and announcements,
- and, perhaps most importantly, establish and maintain a library of public domain, user supplied software.

A preliminary meeting was held at AAAI '86, and a second meeting is being planned for AAAI '87. Over 80% of those who have expressed interest in the users' group are members of the Info-TI-Explorer and Bug-TI-Explorer mailing lists, currently maintained on the SUMEX-AIM 2060. Negotiations with Texas Instruments over the legal ramifications of the user library are in the final stages. The format and procedures of the library have been mapped out, and are currently undergoing peer review. Online copies of the library will be maintained at Texas Instruments, and on the SUMEX-AIM 2060, to facilitate ARPANET access to the software.

There are already many entries ready for the library, most of which have been developed locally. We have maintained the software tools that were produced previously by fixing bugs, making improvements, and porting to new releases. Some of these have remained essentially the same, including:

- The Symbolics 36xx to Explorer compatibility package
- The Source Code Controller (was known as the System Manager)
- Imagen Via TCP (was Net Imagen)
- Finger Via TCP (was TCP Finger)
- Vertically Ordered Menu Columns
- General Named Structure Message Handler
- DEFSTRUCT Type Checking
- Batch Processor
- Choice Facility Enhancements (was Choose Variable Values Macros)
- Backup To File System (was FS To FS Backup)

Many of the tools have been enhanced or newly written this year, including:

- A number of pieces that allow the user to exploit a "desk top" usage metaphor, where several applications can be active, or semi-active at once, with the interaction area, or "window" of each application potentially overlapping others. These pieces include:
    - WINDOW-MANAGER-SYSTEM-MENU provides a replacement to the standard system menu that allows for easy manipulation of the placement and shape of windows on the screen, as well as other common display management operations.
    - RUBBER-BAND-RECTANGLES which allows easy, precise specification of a rectangle on the screen by providing a constantly