III.A.2.3. Core AI Research

1 - Rationale

Artificial Intelligence (AI) methods are particularly appropriate for aiding in the management and application of knowledge because they apply to information represented symbolically, as well as numerically, and to reasoning with judgmental rules as well as logical ones. They have been focused on medical and biological problems for well over a decade with considerable success. This is because, of all the computing methods known, AI methods are the only ones that deal explicitly with symbolic information and problem solving and with knowledge that is heuristic (experiential) as well as factual.

Expert systems are one important class of applications of AI to complex problems -- in medicine, science, engineering, and elsewhere. An expert system is one whose performance level rivals that of an human expert because it has extensive domain knowledge (usually derived from an human expert); it can reason about its knowledge to solve difficult problems in the domain; it can explain its line of reasoning much as an human expert can; and it is flexible enough to incorporate new knowledge without reprogramming. Expert Systems draw on the current stock of ideas in AI, for example, about representing and using knowledge. They are adequate for capturing problem-solving expertise for many bounded problem areas. Numerous high-performance, expert systems have resulted from this work in such diverse fields as analytical chemistry, medical diagnosis, cancer chemotherapy management, VLSI design, machine fault diagnosis, and molecular biology. Some of these programs rival human experts in solving problems in particular domains and some are being adapted for commercial use. Other projects have developed generalized software tools for representing and utilizing knowledge (e.g., EMYCIN, UNITS, AGE, MRS, BB1, and GLisp) as well as comprehensive publications such as the three-volume *Handbook of Artificial Intelligence* and books summarizing lessons learned in the DENDRAL and MYCIN research projects.

There is considerable power in the current stock of techniques, as exemplified by the rate of transfer of ideas from the research laboratory to commercial practice. But we also believe that today's technology needs to be augmented to deal with the complexity of medical information processing.

Our core research goals, as outlined in the next section, are to analyze the limitations of current techniques and to investigate the nature of methods for overcoming them. Long-term success of computer-based aids in medicine and biology depend on improving the programming methods available for representing and using domain knowledge. That knowledge is inherently complex: it contains mixtures of symbolic and numeric facts and relations, many of them uncertain; it contains knowledge at different levels of abstraction and in seemingly inconsistent frameworks; and it links examples and exception clauses with rules of thumb as well as with theoretical principles. Current techniques have been successful only insofar as they severely limit this complexity. As the applications become more far-reaching, computer programs will have to deal more effectively with richer expressions and much more voluminous amounts of knowledge.

Expert systems are being developed that impact nearly every field of human endeavor: medicine, manufacturing, financial services, diagnosis of machinery, geology, molecular biology and structural design, to name a few. Each new instance is a confirmation of the hypothesis that knowledge is power. In each system, expert level problem-solving performance is obtained by using relatively simple and uniform reasoning methods which access an extensive body of domain knowledge. The ability of these systems lies not in their superior reasoning capabilities but in the

specific concepts, facts, methods, models, etc. that can be brought to bear on the problem. The *knowledge is power* hypothesis has received so much confirmation that we now assert it as the *knowledge principle.* A corollary to the Knowledge Principle is that significant improvements in the power of knowledge-based systems will be derived primarily from the ability to access large amounts of knowledge.

During the past year we have begun to explore the design and use of very large knowledge bases. In the last twenty years we have learned how to build intelligent programs that perform at a high level of competence on specialized tasks within narrowly defined domains. These programs traditionally access small to modest-sized knowledge bases specialized to the prescribed task. In contrast, we have started on a long-range research effort that will result in a *large, multi-use knowledge base* (LMKB).

We believe construction of a LMKB is an essential step toward resolving two fundamental problems plaguing the current generation of expert systems. The first is *brittleness*: current systems can exhibit only a very narrow range of expert behavior, and their performance falls off precipitously at the limits of their expertise. The second problem is *over-specialization*: a knowledge base constructed to support of one type of expert task (e.g., diagnosis) cannot be used to support other types of tasks (e.g., design).

Our hypothesis is that both the problems of brittleness and over-specialization can be addressed by constructing large, multi-use knowledge bases. A LMKB would 1) encode domain knowledge in greater depth and breadth than required for any specific task, 2) encode knowledge that cuts across many domains of expertise, and 3) serve as a core repository of knowledge to be accessed by large numbers of specific applications.

This report documents progress on the basic or core research activities within the Knowledge Systems Laboratory (KSL), funded in part under the SUMEX resource as well as by other federal and industrial sources. This work explores a broad range of basic research ideas in many application settings, all of which contribute in the long term to improved knowledge based systems in biomedicine.

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 in PROTEAN and BB1, with its explicit representation of control knowledge (see the summary of Blackboard Architectures below). In particular, we have advanced our methods for representation of geometric problem solving knowledge in PROTEAN and PEAKS (see PROTEAN section of this report.) We have developed an application in a new domain of diagnosis and correction of errors in a linear accelerator beam line, the ABLE project. In this we have explored issues of representation of diagnosis expertise, and have developed a method that incorporates a numerical simulator of a model system with an expert system (see discussion under Knowledge Acquisition and Learning below.) In addition, we continued research on NEOMYCIN which has a component for using a flexible, rich representation of control knowledge to facilitate modeling of problem solving at the strategic level as well as at the tactical level.

[See KSL technical reports KSL-87-58 and KSL-87-62.]

2.2 - Blackboard Architectures and Control

How can we design flexible control structures for powerful problem solving programs? How can we use these structures effectively in many problem domains? How can we represent processes and reason about their behavior, and perform intelligent actions under *real-time* requirements?

We have continued to develop the BB1 blackboard architecture for systems that reason about -- control, explain, and learn about -- their own actions. In the last year, we have significantly extended the system-building support and run-time capabilities of the BB1 system. These extensions include (a) declarative representation of large bodies of factual and heuristic knowledge; (b) integration of multiple reasoning skills in a single system; (c) dynamic control under real-time constraints. We have also implemented the following application-independent components: (a) declarative representation for device structure, function, faults, and repairs; (b) reasoning modules for associative and model-based diagnosis; and (c) an asynchronous communications interface.

During the past year, we began work on an advisory system, called BB-ICU (see also the separate collaborative project report on Page 129), to support patient monitoring in a surgical intensive-care unit (SICU). Briefly, intensive care patients are critically ill individuals who require life-support devices, such as respirators or dialysis machines, to perform some of their vital functions. During their stay in the intensive care unit, patients are monitored closely and gradually weaned from life-support devices in coordination with their changing physiological status and other therapeutic interventions.

We began by visiting the SICU at the Palo Alto VAMC to observe monitoring procedures and operations. We worked with Dr. Adam Seiver to enumerate and characterize component intensive-care monitoring tasks and to delineate the space of relevant knowledge. We developed an ontology and representation scheme for important categories of knowledge (e.g., anatomy, physiology, pathology, therapy) and assessed our approach by implementing a small amount of knowledge in each category. We then enumerated key architectural requirements for BB-ICU and identified those not met in existing AI architectures.

During the fall of 1987, we elaborated our initial ideas in the context of more focused design and implementation activities. Exploiting and extending our BB1 architecture [KSL-84-16, KSL-88-22], we developed: (a) an asynchronous I/O subsystem to provide integrated and asynchronous perception, action, and cognition; (b) an intelligent I/O mediator to translate, interpret, and filter low-level data on behalf of the application system; and (c) an I/O manager to coordinate the mediator's activities with an application system's dynamic attentional focus. Working within our BB* knowledge representation environment [KSL 86-38], we implemented representations of the anatomy, physiology, and pathology of the respiratory system as instances of corresponding elements of a generic flow system. We developed reasoning components for continuous data interpretation and associative diagnosis of observed symptoms. We also developed reasoning components that instantiate generic models, such as the flow system model, to explain the causal relations underlying associative diagnoses or to hypothesize plausible diagnoses in the absence of associative knowledge. We demonstrated the application of the reasoning components and respiratory knowledge to interpret, diagnose, and explain respiratory data of the sort continuously monitored in the intensive-care unit. BB-ICU Demo-1 comprises independently implemented versions of each of these system components. BB-ICU Demo-2, which we completed in April, integrates these system components.

Some of this work is reported in recent technical reports [KSL 87-31, KSL 87-67, KSL 88-20, KSL 88-22). Other reports are in preparation. In addition, we have given talks describing this work at: the Carnegie-Mellon University Symposium on Architectures for Intelligence, Boeing Computer Services in Seattle, Wa., Advanced Decision Systems in Mountain View, Ca., and the DARPA Planning Workshop in Austin, Tx.

2.3 - Advanced Architectures

Many applications, such as process planning and control, maintenance, troubleshooting, environmental control, and crisis management require knowledge-based systems that can cope with large amounts of data and that produce responses in real-time. The current hardware and software architectures for knowledge-based systems cannot support such requirements. The most promising approach for achieving orders of magnitude improvement in the quantitative performance of knowledge-based systems is by exploiting concurrency on multiprocessor systems. Based on near-term projections for integrated circuit technologies, it is clear that highly parallel multiprocessor computers consisting of 100's to 1000's of processors and realizing a variety of concurrent architectures can be built. The major issue is whether such computers can be effectively used to enhance the performance of knowledge-based systems. Since 1985, the Knowledge Systems Laboratory at Stanford University has been investigating this issue.

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 was concluded in July 1987, we 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.

Our research methodology is:

- Select specific knowledge-based system applications, primarily signal understanding applications.
- Encode these applications following various proposed concurrent software models.
- Evaluate the qualitative and quantitative performance of the applications running on simulated multiprocessor machines with respect to varying hardware parameters, for example, number of processors and communication protocols, and varying software organizations, for example, degree of control centralization.

In the following discussion, we present the major components of our project, and for each component we describe its current status.

2.3.1 - SIMPLE/CARE Multiprocessor Simulation System

Simulation of systems at an architectural level can offer an effective way to study critical design choices if (1) the performance of the simulator is adequate to examine designs executing significant code bodies -- not just toy problems or small application fragments, (2) the details of the simulation include the critical details of the design, (3) the view of the design presented by the simulator instrumentation leads to useful insights on potential problems with the design, and (4) there is enough flexibility in the simulation system so that the asking of unplanned questions is not suppressed by the weight of the mechanics involved in making changes either in the design or its measurement.

SIMPLE/CARE is a simulation system which satisfies these requirements. It forms the foundation for our empirical investigations of software architectures and hardware system architectures for concurrent knowledge-based systems. SIMPLE is a CAD (Computer Aided Design) system for hierarchical, multiple level specification of computer architectures and includes an associated mixed-mode, event-based simulator. CARE is a parameterized, multiprocessor array emulation specified in SIMPLE's specification languages and running on SIMPLE's simulator. Our simulation system is in use by several research groups at Stanford, and it has been ported to several external sites including NASA Ames Research Center. A tutorial was held in January, attended by representatives from the DoD, NASA and Boeing, which described the CARE/SIMPLE system, as well as the LAMINA programming interface (see below). The attendees received instruction in use of the system for making measurements of the performance of various simulated multiprocessor applications.

The SIMPLE/CARE system is currently implemented in ZetaLisp and executes on Texas Instrument Explorer and Symbolics 3600-class Lisp workstations. We have recently started a reimplementation of the system in Common Lisp. One of our proposed objectives during the coming year is to complete this reimplementation. A Common Lisp version of SIMPLE/CARE will make it portable to a wide variety of computer systems including Sun and MicroVAX workstations. This development will necessarily be an ongoing task as Common Lisp standards, in particular, window standards, evolve and as the inevitable commercial reinterpretations of standards emerge.

The SIMPLE design specification system has design operators for automatically generating array type multiprocessor architectures from a "unit cell" specification. There is considerable interest, both at Stanford and elsewhere, in using the system

to specify and simulate other types of multiprocessor architectures. A second proposed objective is to augment SIMPLE's design operators with recursive operators for the generation of architectures using, for example, hierarchical busses or recursive interconnection nets such as Omega nets.

2.3.2 - LAMINA Programming Interface

LAMINA provides extensions to Lisp for studying expressed concurrency in functional programming, object oriented, and shared variable models of concurrent computation. The implementation of the support for all three computational models is based on the common notion of a stream, a data type which can be used to express pipelined operations by representing the promise of a (potentially infinite) sequence of values. LAMINA also provides system support for the management of software pipelines and dynamic structure creation, relocation, and reclamation in a multiprocessor, multi-address-space system.

Algorithms and applications written in LAMINA may be run on the SIMPLE/CARE simulation system in order to study their execution on alternative multiprocessor architectures.

The development of LAMINA was essentially completed during the past year, and the software is now reasonable stable. In order to make LAMINA available in the community, we intend to port it to Common Lisp. We also expect that the application research will motivate various extensions to the LAMINA programming interface.

2.3.3 - Poligon Problem Solving Framework

Poligon [KSL 86-19, KSL 88-04] is a framework for the development of Blackboard-like applications on a (simulated) multiprocessor. It consists of:

- A compiler, which compiles a high-level description of the Blackboard's structure and the Knowledge to be applied by the system, to run on a distributed memory multiprocessor.
- A run-time system which provides a debugging and testing environment for Poligon programs as well as run-time support.

Both the compiler and the run-time system are thoroughly integrated with the program development environment of TI Lisp machines, the machine on which the execution of Poligon programs are simulated.

Serial Blackboard Systems are implemented with the Nodes being represented as records on the Blackboard. The Knowledge is encoded in Knowledge Sources. These are typically compiled into procedures which are invoked by the Blackboard System's kernel. There is some form of scheduler for the Knowledge, which invokes one Knowledge Source after another. The Blackboard and the Knowledge Base both share the same address space, though they are functionally distinct. Knowledge Sources are "invoked" (executed) as a result of changes in the Blackboard placing that change event in a queue used by the scheduler. The scheduler repeatedly picks a Knowledge Source which is interested in the type of event at the end of the queue.

Experiments with Poligon are by no means complete, but we have learned a number of lessons thus far. Some of these lessons are enumerated below.

- It is very hard to write any program which implements either a framework, such as Poligon or an application such as those which have been mounted on Poligon. This is due largely to asynchronous side effects. A system with better formal properties would be less error prone in this respect but might well make much less efficient use of the hardware. These difficulties could also be caused by an insufficiency of mechanisms to control coherency in Poligon.

- In order to produce a reliable program it is necessary to write code which makes no assumptions about anything that any other part of the system might be doing. Failure to do so results in brittle systems.

- In order to achieve a coherent solution it was found to be necessary to develop a number of programming methodologies. For example, the creation of blackboard Nodes is tricky. Because each element is likely to represent some real-world object, it is important either to provide a mechanism for resolving the conflict caused by multiple asynchronous requests to create an element that represents the same thing or to provide a mechanism for managing the creation of Nodes. Poligon opts for the latter approach.

2.3.4 - CAGE Problem Solving Framework

CAGE [KSL 86-41, KSL 88-02] is a framework for building and executing applications as a concurrent blackboard system. CAGE is based on the AGE [KSL 80-29] serial blackboard framework. It includes mechanisms for the concurrent execution of knowledge sources, rules and parts of rules. The CAGE user has complete control over which of these mechanisms are used. CAGE is designed to execute on a shared-memory, multiprocessor system with tens to hundreds of processors. It is implemented using Qlisp, a concurrent dialect of Lisp designed for multiprocessors with a single, shared address space. CAGE currently executes on a shared-memory variant of CARE simulated using the SIMPLE simulation system.

We are nearing completion of a series of end-to-end experiments for evaluating the utility and performance of the CAGE concurrent blackboard framework. During the coming year we intend to complete these experiments and disseminate the results.

2.3.5 - CAGE, Poligon and LAMINA Comparative Experiments

During the past two years we have been developing application software and machine architecture models to support a series of end-to-end experiments comparing various concurrent programming systems for knowledge-based applications. The goals of these experiments are to:

- Obtain quantitative comparisons of the performance of the programming systems.

- Gain insights into how different concurrent programming models lead to different (or similar) application decomposition and organization.

- Force the refinement of the concurrent programming systems so as to better support application development.

- Gain insights into the ease or difficulty of writing application code in each of the programming systems.

The common application for these experiments is Elint [KSL 86-69], a real-time, knowledge-based system for integrating pre-processed, passively acquired radar emissions from aircraft. This Elint application has been implemented in three different concurrent programming systems: LAMINA, Poligon and CAGE.

Each of the implemented applications are executed and evaluated using various input data sets and varying numbers of processors.

Application code written in either LAMINA or Poligon compiles to code which executes on the CARE architecture. CAGE, however, is targeted toward a single address space, shared variable multiprocessor architecture. CAGE is implemented in QLisp, a concurrent Lisp for shared variable multiprocessors. To support CAGE we had to develop a multiprocessor "blackboard machine" variant of CARE. This blackboard machine models a shared variable architecture and includes the mechanisms and instruments necessary to manage and study memory contention. The architecture implements the blackboard and the control data structures in global, shared memory. It directly supports the CAGE system and application code written in QLisp.

During the past year we have:

- Completed the implementation of the the Elint application in each of the three concurrent programming systems.
- Completed the development of the blackboard machine variant of CARE.
- Developed an experiment plan for the comparative studies.
- Developed a new measure of speedup as a function of the number of processors in a multiprocessor system. This measure is useful for evaluating system performance of real time applications and is based on the concept of maximum sustainable input data rate.
- Completed the first set of experiments for each of the three programming systems.

2.3.6 - The AIRTRAC Application

AIRTRAC [KSL 86-20] is the primary application driving our development of concurrent knowledge-based system programming methodologies. Also, it is one of the basic applications used for our multiprocessor architecture performance experiments. AIRTRAC is a knowledge-based signal interpretation and information fusion system. The system attempts to identify, track, and predict the future behavior of aircraft. In particular, it attempts to recognize aircraft which might be engaged in covert activity, for example, smuggling. The inputs to AIRTRAC are periodic radar tracking system reports, a priori, filed flight plans for some aircraft, and occasional intelligence reports about suspected covert activity.

AIRTRAC is designed to be sufficiently complex and realistic to adequately test various ideas about concurrent problem solving on multiprocessor machine architectures. The AIRTRAC application involves continuous input data streams, typical of real-time signal interpretation problems. Such problems often require a level of computational power two to three orders of magnitude beyond what is currently available. Moreover, the application uses data-driven, expectation-driven and model-driven styles of reasoning. These reasoning styles encompass a wide range of paradigms in artificial intelligence.

The AIRTRAC Data Association Module and associated experiments were completed as of summer, 1987 [KSL 87-34]. The experiments were performed using the SIMPLE/CARE multiprocessor simulation system. They demonstrated that almost linear speedup as a function of the number of processors can be achieved (at least up to 100 processors) for a periodic data-driven knowledge-based system such as the Data Association Module.

During the past year, the design and knowledge acquisition for the Path Association Module was completed. Over one half of the LAMINA code for this module has been implemented and debugged.

The completed AIRTRAC application will provide an end-to-end example of a concurrent, knowledge-based signal interpretation system. It will demonstrate the benefits and costs of implementing and executing such systems on multiprocessor architectures. Also, the application is sufficiently complex that it will serve as important test case for evaluating multiprocessor architectures for knowledge-based systems and "tuning" the engineering parameters for such systems.

2.4 - Knowledge Acquisition and Machine Learning

Our research in machine learning has focused on several distinct problem domains including medical (NEOMYCIN/HERACLES), physics (ABLE), 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).

2.4.1 - Learning by Chunking

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. Another emphasis in SOAR is Explanation-Based Learning, a powerful technique that generalizes concepts learned from examples. In this past year, SOAR has demonstrated acquisition of diagnostic problem solving knowledge (similar to that in NEOMYCIN), learning from multiple examples and from analogy, and learning attribute-value information to acquire features of a single object incrementally, reusing known objects as values of attributes. The work on SOAR is continuing under Dr. Paul Rosenbloom at ISI in Los Angeles.

2.4.2 - Inductive Rule Learning

In previous reports, we discussed the work of Buchanan, et al. on incremental learning process from examples with the rule-learning system RL (described in the 1986 SUMEX report). This work has continued, and has been applied to the domain of linear accelerator physics (see below). Results from the RL research indicate 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.

In the paper Simulation-Assisted Inductive Learning, to be presented at the 1988 AAAI conference in Minnesota, Buchanan, Clearwater, et al. describe the work done as a collaborative effort between the Rule Learner project and the Automated Beam

Line Experiment project. The focus of this work is to show how RL can be used in a real-world domain with sparse data by working with a simulator which numerically models the domain. The domain is the classification and location of faults in a particle accelerator. This study demonstrates the effectiveness of RL even in this noisy, numerical domain. Some problems in generating the best examples from which to learn rules and how to learn the best rules from a given set of examples are explored. In addition, methods of weighing the evidence when several rules fire are being investigated.

2.4.3 - Learning Apprentice

In addition, we continued several investigations of methods for building knowledge bases for knowledge-based programs. Knowledge engineering remains the "standard" method of building a knowledge base for commercial systems, so we have investigated ways of making that more efficient. Technical reports KSL 87-58 and KSL 87-62 describe some of the results, pertaining mostly to the principles of starting with a sound problem solving strategy and of exploratory programming. Reports KSL 87-60 and KSL 87-67 describe work on learning apprentice methods.

In last year's SUMEX report, we reported results on the ODYSSEUS apprenticeship learning program, described by Wilkins in KSL 86-63, which is 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. The system 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 [KSL 86-62]. In the last year we began the creation of a case library from medical records for the NEOMYCIN domain. This library is essential for apprenticeship learning in ODYSSEUS in three ways. First, experiments have shown that the existing knowledge bases are too impoverished to follow the reasoning of an expert, and a case library will allow an induction system to automatically expand the domain knowledge to overcome this limitation. Second, when the learning system fails to explain an action of a student or expert correctly, the critic component of the system generates thousands of conjectures. To filter these requires a case library. Finally, the case library will allow us to demonstrate that apprenticeship learning can improve the performance of an expert system.

2.5 - Pragmatic Approaches to Reasoning Under Uncertainty

The goal of this project is to investigate pragmatic approaches to computer-based probabilistic reasoning systems. In the past, artificial intelligence researchers have often avoided probability theory for reasoning with uncertainty because of the perception that the application of probability is invariably associated with a commitment to intractable algorithms and an inordinate amount of knowledge acquisition time. The development of efficient probability inference and assessment techniques will allow investigators to apply a theoretically justified theory of belief entailment to complex problems. Specifically, this project seeks to (1) develop

techniques for using knowledge about problem-solving tradeoffs to dynamically optimize the value of computer performance to the user, (2) construct efficient algorithms for probabilistic reasoning, and (3) investigate pragmatic techniques for the elicitation of knowledge from experts.

2.5.1 - Reasoning about inference tradeoffs

Research on reasoning tradeoffs has focused on the time vs. quality-of-result tradeoff in several domains. Additionally, tradeoffs arising with the representation of knowledge and explanation of inference within a probabilistic framework have been identified [KSL 88-13] The growing perspective of this research is that problems traditionally ascribed to knowledge representation, inference, and explanation in probability-based reasoning systems have been encountered because of insufficient attention given to tradeoffs under bounded or varying resources available for engineering, computation, or cognition [KSL 88-13,87-28].

Notable research on inference tradeoffs during this past year has been the implementation of a prototype strategic reasoner for the analysis of tradeoffs. Fundamental issues of control under varying resource limitations were explored with simple sorting algorithms [KSL 88-3]. Some of this work will be reported in an article appearing in the AAAI conference this summer. Other work on inference tradeoffs has focused on applying a similar computational architecture to control the selection of alternative belief-network inference strategies [KSL 87-64].

Work has been carried out on the implementation and characterization of useful classes of probabilistic approximation algorithms that contain explicit tradeoffs. There has been special interest in the development of flexible strategies for reasoning under uncertainty with varying resource limitations. Recent work along these lines has focused on the development of modified versions of a probabilistic inference technique developed by Pearl [KSL 88-27]. The new approach allows the performance of the algorithm to be "gracefully degraded" under resource limitations, through pruning the consideration of terms in accordance with their expected effect on the final answer. Preliminary empirical analysis of the time/accuracy tradeoff for this algorithm has been carried out on a multiply-connected belief network.

2.5.2 - Efficient probabilistic inference algorithms

During the past year, two of our primary goals have been to implement several known probabilistic inference algorithms and begin to test their efficiency. In particular, we have implemented Pearl's algorithm for multiply-connected belief networks and in the process we have gained important insights into the nature of the algorithm [KSL 88-27]. These insights have allowed us to make design choices that yield an efficient implementation the algorithm. In particular, we have designed and developed a fast method for finding and using a cutset of nodes for inference in networks that contain complex loops. We also are working to improve the efficiency of updating belief networks that do not contain loops; our current algorithm is able to update a singly-connected network of propositional variables at a rate of approximately 50 nodes per second on a Macintosh II in Turbo Pascal.

Our implementation of the Pearl algorithm has been successfully tested using a set of benchmarks of varying complexity that we have developed. These benchmarks soon will be used to test comparatively the computational time complexity of several additional inference algorithms. We also have begun a collaboration with a researchers at the University of Aalborg in Denmark who are leading an effort in Europe to develop expert systems based on belief networks. We have already

shared some of our benchmarks with them. In the coming months we anticipate increased collaboration that will include comparison of benchmark timing results and the exchange of algorithms.

The Pearl algorithm that we have implemented will serve as one type of exact algorithm. In addition, we are currently implementing a probabilistic inference algorithm by Lauritzen and Spiegelhalter. The Pearl and Lauritzen/Spiegelhalter algorithms will be our initial set of exact algorithms. We also plan to study approximation and heuristic techniques for probabilistic inference. This is particularly important in light of our proof that exact probabilistic inference using belief networks is NP-hard [KSL 87-27]. We previously implemented a stochastic algorithm and will use this as our initial approximation algorithm. An neural-network algorithm will be used as an initial heuristic algorithm. We anticipate that by the end of the summer we can begin to compare these algorithms on sets of theoretical and real inference problems. These tests will provide important information for designing new algorithms in the coming year.

Several other researchers in the Medical Computer Science Group are currently using our inference algorithms to develop expert systems based on belief networks. Systems that presently are being developed include 1) an intelligent anesthesia monitor, 2) a diagnostic system for the Intensive Care Unit, and 3) a system that assists in evaluating the statistical validity of a clinical drug trial report. These applications have given us practical feedback about the level of inference performance that is necessary in real domains. Although these applications are still in the early stages of development, they suggest that improved inference efficiency will be a critical issue in producing practical expert systems that are based on belief networks.

During the past year we have developed a general knowledge engineering environment called KNET (Knowledge NETwork) on a Macintosh II in MPW Pascal [see Chavez article]. KNET is a flexible graphical interface system for entering a belief network [see Lehmann article] and running cases using a belief network. It provides a general software system foundation from which to experiment with different methods for pragmatic probabilistic reasoning. For example, a key feature is that KNET provides a modular environment in which different inference techniques can be tested. We recently were able quickly to install Pearl's inference algorithm into KNET. We soon plan to have several other inference algorithms running in KNET.

Although the emphasis in the early stages of this work has been on developing an initial set of algorithms, we have also designed several new techniques. One technique uses dynamic programming to solve efficiently a large class of complex probabilistic inference problems. We have not yet implemented and tested this algorithm. However, it appears that for some types of complex belief network topologies this algorithm will be very fast relative to current techniques. Another method that was recently developed in our group allows any belief network algorithm to be used to solve decision problems (i.e., influence diagram problems) [KSL 88-28]. This general method significantly broadens the scope of application of our work on belief network algorithms. Of particular interest, it allows currently available exact, approximation, and heuristic belief network algorithms to be easily adapted to solve decision making problems.

2.5.3 - Probability assessment

During the past year, the majority of the research outlined for the reasoning-by-two assessment method, now called *similarity networks*, was completed. We began by implementing the similarity network approach in Turbo Pascal on the Macintosh II. We then evaluated the knowledge assessment tool by using the program to build a small component of the Pathfinder knowledge base, a module that helps a novice pathologist classify spherical structures seen in a lymph-node tissue section. Using the program, our expert was quickly able to identify the morphologic features relevant to the classification task and was able to specify the dependencies among these features.

Upon using the program for probability assessment, it became clear that a generalization of the similarity network would be useful for reducing the number of assessments required. In particular, it became clear that arbitrary sets of hypotheses should be allowed to be clustered together and labeled as "similar." A pen and pencil approach to eliciting probabilities in this manner was developed and used to assess the probabilities required for the entire lymph-node pathology domain. In assessing the probabilities, it was assumed that all observations are conditionally independent on each hypothesis. The approach was quite successful as it reduced the number of probability assessments required of the expert by a factor of approximately twenty (from 30,000 assessments to roughly 1,500).

Finally, the performance of the knowledge base constructed in this manner was evaluated [KSL 88-38]. In doing so, a new evaluation metric based in decision-theory was developed. The results of the evaluation demonstrated that the performance of the knowledge base was close to that of the expert. However, the results also showed that there is still room for improving the knowledge base through the representation of dependencies among features.

2.5.4 - Collaborations

During the past year we have continued to maintain contact with a number of Stanford faculty who are interested in the research goals of this grant. We have also collaborated with visitors from outside of Stanford. In particular, Prof. Max Henrion visited our group during the Autumn quarter. We continue to communicate frequently with him about our common research interests. We also have benefited from visits by Dr. David Spiegelhalter (a statistician from the Medical Research Council in England), Dr. Stig Andersen (a computer scientist from Aalborg University in Denmark), and Prof. Ross Shachter (a Stanford faculty member on sabbatical at Duke University).

Most of the members of our group were able to attend the recent conference on Influence Diagrams in Berkeley. We have submitted several papers to the AAAI Uncertainty Workshop this year and most of us plan to attend this workshop in August.

III.A.2.4. Core System Development

1 - Introduction

In this section we describe progress on our core system development and work toward a distributed AIM community. In last year's report, we discussed the motivations and plans for our core system development work along four dimensions: 1) the motivation for the shift of the SUMEX-AIM community from a central mainframe-based computing resource model to a largely distributed workstation-based model; 2) the prospects for workstation technology and vendor support for a diverse distributed AIM community; 3) the projected 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. These were expanded still further at a site visit held in August 1987 in response to our request to the National Advisory Research Resources Council to restore the final 2 years of our grant award. Following a special study section review and reconsideration by the Council, our plan and the full 5-year grant award were approved. The review group's concluding recommendations included the following guidelines:

> "Consistent with its charter as a national resource, SUMEX should focus its systems activities on producing a distributed medical research environment that can be easily reproduced at other sites. It should also continue to play the important role it plays today as a repository of systems information and expertise for the medical AI research community, as well as the larger computer science community. However, it should avoid trying to be all things to all people and should focus its attention on a small number of standardized hardware and software configurations. A strong effort should be made to acquire information about related systems activities at other sites and to avoid duplication of effort. These guidelines should be used to establish priorities among the proposed set of system activities and to apply effort appropriately."

The review committee's recommendation was very much in line with our own goals to more sharply focus our development resources and much of our effort over the past year has been devoted to that end. Since our 1980 renewal proposal in which our move to distributed workstation technologies began, we had taken on the development and support of a wide array of systems including mainframes (DEC 2060 and 2020), network servers (2 DEC VAX 11/750 UNIX file servers, a SUN 3/180 UNIX file server, a Xerox file server, 7 network laser printers, Ethernet gateways and TIP's, and ARPANET and TELENET wide-area access systems), and workstations (50 Xerox D-machines, 20 TI Explorers, 6 Symbolics machines, 5 Hewlett-Packard 9836 workstations, and 3 SUN 3/75 workstations). Whereas we cannot drop support for these systems irresponsibly, we resolved to pick a much more limited environment on which to focus our long-term systems development efforts and to phase out support for the other systems as quickly as possible.

In summary, we have chosen Apple Macintosh II workstations as the general computing environment for researchers and staff, TI Explorer Lisp machines (including the microExplorer Macintosh coprocessor) as the near-term high-performance Lisp research environment, and a SUN-4 as the central system network server (wide- and local-area network interfaces, file services, printing services, etc.).

2 - Distributed System Evaluation and Selection

2.1 - Design Goals

In planning for AIM community computing needs for the next few years to replace and upgrade the powerful and easy-to-use general computing tools and network services of the 2060, several goals were identified:

- The work environment should be modern and combine graphics, pointing, and traditional keyboard modalities of interaction, as it is expected to be the primary work environment for some years to come.
- The system should support the most powerful AI research and Lisp development environment available today, possibly involving special-purpose hardware.
- The system should support small-to-medium-size AI and Lisp-based research work without requiring special hardware.
- The cost per person should be low enough as to permit putting a machine on or near every desk and to consider the system as a potential AI delivery environment.
- The system should integrate well into a heterogeneous computing environment typical of AIM research work.
- The system should be capable of editing, organizing, and printing large documents, such as theses and books.
- The system should be capable of generating and editing state-of the art graphics.
- The design should be incrementally extendable and augmentable as new hardware and software technologies appear and as the number of users fluctuates.
- The design should be simple enough as to refocus our systems work on a smaller number of machines and cost-effective enough as to be replicable at smaller AIM sites that wish to benefit from our experience.
- The design should permit easy data sharing and exchange with collaborators at other sites and within Stanford University.

In addition to user-related computing tools, other heavily-used network services traditionally provided by the shared 2060, must be replaced. These include wide-area network access (ARPANET and TELENET), electronic mail (transmission/routing, reception, and user access), community bboards, file service, and print spooling.

2.2 - Evaluation Results

We examined many potential configurations before deciding on the solution involving Mac II's, microExplorer's, and the SUN-4 . Many of the considerations were technical in terms of the tools and services provided by the systems and many had to do with user preferences for interface style and environment.

Timesharing machines were eliminated for their lack of modern interactive productivity tools. (The many reasons for the trend from timesharing to workstations

have been discussed in previous annual reports. The pressures behind this trend have grown stronger with time.)

Xerox lisp machines were eliminated by virtue of their uniqueness and the questionable future of the hardware product line. Stand-alone TI (and other similar) Lisp machines were eliminated by virtue of their uniqueness and high cost and lack of general computing tools for mail, document preparation, etc.

Sun workstations were eliminated by virtue of their relatively higher cost and engineering orientation and their dependence on the UNIX user interface which received almost uniformly negative comment in a KSL user questionnaire about computing environment preferences.

IBM PC's were eliminated because of current limitations in their primitive operating system, window system, and interface style, when compared with the Macintosh.

Of course, this kind of evaluation is very complex and the above reflects only a summary of key issues. There are many reasons for or against any of the above machines not fully enumerated here. In the end, we chose the Apple Macintosh II primarily because of the following considerations:

- the Mac has a powerful, intuitive, and consistently applied icon-based user interface that facilitates wide use and effectiveness.
- The Mac II is a powerful machine (Motorola 68020-based) with an open architecture that provides long-term configuration flexibility (e.g., for color, coprocessing units like the microExplorer, memory, i/o devices, etc.).
- The Macintosh is popular and is used by a growing number of our collaborators. Many members of the AIM community specifically endorsed the Macintosh as their machine of choice and many already have Macintoshes at home.
- Apple gives educational institutions a substantial discount. This and the low price of third-party disks meant we could put a Mac on almost every desk.
- There is a large variety of third-party hardware and software available. Competition and volume mean lower prices; especially when compared to Sun or DEC third-party offerings.
- Texas Instruments announced the *microExplorer*, a board-level product which gives the user a custom VLSI lisp machine inside his Mac, for a fraction of the cost of TI's stand-alone *Explorer* workstation product.

To replace the network service functions of the SUMEX-AIM 2060, we chose a Sun 4/280 system. We investigated competitive systems, for example, the DEC VAX and Pyramid Technology 9000 series product line. Configurations are available from 3.5 to 25 MIPS and with individual I/O channel speeds up to 11 MBytes/Sec. However, we decided that the SUN-4 was more cost effective, had more popularity at Stanford and at other universities, and offered outstanding research network service software. This configuration is at the beginning of its product cycle and can be expected to serve for many years to come.

The SUN 4/280 approach was made even more attractive by the opportunity to have it equipped with state-of-the-art 900 megabyte disks. This option was presented to us after a review of second-source disks indicated that SUN's then offered Fujitsu Double Eagle units (575 megabytes) were not the optimum in cost-effectiveness.

2.3 - Configuration

The working plan we eventually settled on called for a Sun 4/280, 69 Macintosh II workstations, 10 TI microExplorer upgrades, 7 Kinetics FastPath gateways, and a combination of upgrade packages yielding 2 20 page/minute PostScript laser printers.

The Sun 4/280 is configured with 32 megabytes of memory and 1.8 gigabytes of disk space. By choosing this package we were able to purchase the system for 40% off list price. An ARPANET interface for the Sun server is available and will be purchased in the near future. This will make the new server more readily available to AIM users outside Stanford. This server is currently up and undergoing test and configuration.

By a special arrangement, we purchased 66 Macintosh II computers directly from Apple, each with 1 MB of memory (no disk, no display). At this writing, almost all of the the Macintoshes have been installed. Three units have proven defective and are being repaired under warranty.

A package of 10 microExplorer upgrades was ordered from Texas Instruments. They arrived during the preparation of this report and will be installed shortly. (The features of the microExplorer are described in the *Explorers* section of this report.)

10 100 MB disks were purchased from *Rodime* and installed in the Macintoshes receiving the microExplorer upgrades. (A large paging disk is required by the microExplorer's use of virtual memory.) The balance of the Macintoshes were outfitted with 20 MB disks, also purchased from Rodime. The choice of Rodime disks was suggested by their low price (30% less than Apple's higher-education discount price) and long warranty (12 months). All disks have been installed and so far none have failed. Small disks were deemed sufficient, as users are encouraged to keep their files on the Sun file server (using the AppleShare filing protocol) for reasons of data backup and security.

Our experience with large displays on other workstations suggested that we wanted the largest displays the market could offer at a reasonable price. We chose 33 *Moniterm* 19" displays and 33 *Moniterm* 24" displays, which we were able to obtain in a package at 40% off list price. Our experience with other third-party Macintosh displays told us that a resolution of no greater than 72 dots/inch is easiest on the eyes. Both of the models we purchased conform to this resolution.

Also from our previous experience with Macintoshes, we knew that many applications require more than 1 MB of memory. In our initial purchase, we specified 12 4 MB memory upgrades. These were installed in 12 Macintoshes used primarily by staff and student developers of Mac software. The original 1 MB of memory was removed from each of these machines and added to 12 other machines, making a total of 2 MB in each of those machines. (10 of the 12 were the microExplorers.) We have already concluded that our memory demands require that we do the same with the remaining 1 MB Mac's. Apple's memory orders are now backlogged 5 months, so we have ordered from National Semiconductor instead.

To network the majority of the Macintoshes in the near term, we chose the *Farallon PhoneNet* system which enabled us to reuse terminal wiring we had previously installed in all offices and student areas. In addition to this reason, we chose PhoneNet over Apple's *LocalTalk* wiring system because PhoneNet permits nets 3-4 times larger (by reason of different shielding and impedance characteristics).

To connect the PhoneNet networks to the SUMEX-AIM Ethernet, we chose to install 7 *Kinetics FastPath* gateways. The FastPath is a commercial spin off resulting from

the SUMEX work on the *SEAGATE* gateway. Owing to an earlier royalty payment agreement with Kinetics, we were able to procure the FastPath gateways at no cost. The number of gateways was chosen primarily because of the limited throughput characteristics of PhoneNet (230.4 Kb), but also to provide for hardware redundancy.

To provide printing services for this number of Macintoshes, it was necessary to procure additional PostScript printers. Although Apple offers a substantial discount on its LaserWriter products to its higher-education customers, the 8 page per minute maximum speed (typically less) was deemed too low for our demand printing needs. (Other complaints about the design of the LaserWriter concerned the small paper tray and toner capacity.) As noted in the *Printing Services* section of this report, we obtained a no-cost PostScript upgrade for our 20 page per minute *Imagen 3320* printer in consideration of our having beta tested the upgrade product. Our experience with the basic 3320 product over the past year has been positive. Our positive reaction to the PostScript upgrade convinced us that duplicating the configuration in our other offices was a good idea. We were able to do this by upgrading an Imagen 12/300 to a 3320 PostScript product at a 30% discount. The 3320 is appealing mostly for its print quality, speed, and minimal maintenance requirements. It holds a ream of paper and can print on 11" x 17" paper.

2.4 - More Details about the Transition Plan

Having selected the Macintosh and microExplorer systems for our work, many additional decisions remained to select, configure, and integrate the routine computing environments for our users. The following summarizes this work.

Text Processing - Editing

There are many criteria for a system text editor including:

- Easy to learn
- Available from various contexts so that similar techniques can be used in editing mail, reports, and code.
- Powerful manipulation facilities allowing structures such as words, lines, paragraphs, pages, expressions, code blocks, etc. to be selected, moved, copied, reformatted, transposed, etc.
- Interchange ability allowing at least plain text to be imported from other systems and exported back to them.
- Extensibility in the form of keystroke macros and, ideally, customization libraries allowing us to write packages that make the editor "understand" a new kind of document structure.

Most of the commercially available Macintosh editors are targeted to desk-top publishing and so are fairly easy to use, but have manipulation facilities only for words, lines, paragraphs, sections, and pages. They are sadly lacking in understanding of other types of document structures such as programming languages or electronic messages. These editors typically offer interchange of "plain text" ASCII-only documents. They uniformly offer negligible extensibility.

Of the systems we've looked at, MicroSoft Word has proven most useful of those currently on the market, so we are using it in the meantime. Early demonstrations and tests of FullWrite Professional (marketed by Ashton-Tate) indicate that it may be

superior to Word but its commercial release is just taking place. The non-commercial GnuEmacs might offer a complimentary solution as it is an outgrowth of the Emacs editor widely used in the AIM community. It offers familiarity and powerful extensibility, but it does not offer the easy-to-use interface and multi-font display expected on the Macintosh, and having two different editors would complicate matters.

In the coming year we plan to:

- Track new editor programs, seeking one that better meets our criteria
- Talk with editor vendors to encourage the addition of the desired features
- Further investigate GnuEmacs for the Macintosh II

Text Processing - Aids

Most of the commercial text processing (TP) packages that we've looked at have built-in spelling checkers, sorting, hyphenation, forms filling, etc.

Text Processing - Graphics

MacDraw, MacPaint, and the other Macintosh drawing programs offer state-of-the-art TP graphics capabilities. Pictures from these programs can almost always be integrated into a document being prepared with one of the commercial TP systems.

Text Processing - Formatting

Most of the commercial TP systems are "What You See Is What You Get (WYSIWYG) editors, giving an on-screen representation of the final document during editing. The formatting quality and style control is quite good in the better systems, but we have found that some documents still require the extremely precise control offered by TeX and so are also using it, with it's "compile the manuscript with embedded commands" style interface.

Text Processing - Bibliographic References

Another major shortcoming of the TP systems is the lack of automatic bibliographic reference generation and formatting. In writing scientific papers it is important to cite relevant work, and we have found it extremely useful to be able to extract the significant information from a large bibliographic database by placing a reference key where the citation should appear in the text. We are pursuing TP vendors in the hopes that they will implement this facility as well as investigating development of an auxiliary program to handle bibliography generation.

Printing

Macintosh printing is fairly well developed in that most programs utilize the system-defined routines to print. We have installed PostScript on Imagen printers as well as Apple LaserWriters and are in the process of bringing up a spooling system on the file server.

Help Facilities

Most programs have built-in help on the Macintosh, as well as reasonably consistent interfaces, but this is not enough. We find that users are still confused so we are undertaking to produce short introductory documents to help users get started, and to point them in the right direction. We will investigate using HyperCard to organize this data.

System Information

As the system configuration has gelled we have begun thinking more and more about tools and protocols for getting information about the status of the overall system to the users and maintainers. We will need to address network loading, user location, resource usage (file space, printing, computing cycles), and status information for individual elements of the distributed computing environment. We also need access to personnel information, bulletin boards, and other shared databases.

Interpersonal Communication

See the section on electronic mail development

Systems Building Tools

We are developing expertise in the Macintosh Programmer's Workbench (MPW) environment, including with C and PASCAL, and HyperCard. We are also tracking Allegro Common Lisp, and Neuron Data's NExpert.

Filing

We plan to stay with the strategy of a few centrally located file servers, but the local disks on the Mac's complicate the system. The foremost concern is data backup. We have sketched out a design that would automatically copy new versions of documents (files) created on the Macintosh to a reliable file server (i.e., one that is backed up to tape). This backup program will allow for exclusion of some files (e.g., temporary files) and will make an effort to not have multiple copies of the same file on the server.

Also of significance to users who keep files on the Mac rather than a file server is the resultant inaccessibility from other computers. The proposed backup scheme would alleviate this problem as well. A full UNIX-based file backup and archival system is under consideration for the servers.

3 - R&D Task Plan -- Update and Progress

In the presentation to the site visit review team and Council, we layed out a detailed plan for our developments (see Figure 1). The following summarizes our pruning and reprioritization of those goals, based on the Council review, and progress this past year. In general, wherever we showed parallel developments to maintain capabilities among Xerox, Symbolics, TI, and other workstations, we have restricted our efforts to the Mac and Explorer environments, in accord with the Council recommendation for a focus of effort. While we continue to stay abreast of new workstation hardware and software, we have concentrated our system development work in the following areas for the Mac/Explorer environments. Progress in some areas has been limited by the reductions in systems manpower necessitated by NIH cuts in our award funding.

3.1 - Remote Workstation Access

- **TIP TCP-IP support**: We now have TCP-IP software running in our EtherTIPs.
- **Workstation TCP-IP access**: Each vendor has supplied the requisite software.
- **TELENET X.25/TCP-IP Ethernet Gateway**: The DEVELCON gateway has been installed and is used by the SUMEX-AIM community for TELENET access to the DEC 2060.
- **TELENET TELNET access (TCP-IP)**: The DEVELCON gateway is a bi-directional protocol translating gateway between X.25 and TCP-IP, and thus, fulfills this requirement.

3.2 - Remote Virtual Graphics

- **X Common Lisp client & server**: A Common Lisp X (CLX) client has been released for TI Explorers, SUN Workstations, and Symbolics. An alpha release of a CLX server is expected from TI this Summer. Since Xerox is moving its lisp environment to SUN workstations, and CLX runs on SUNs, we are not going to port CLX to Xerox D-machines. X runs under Apple UNIX (AUX) on MAC II's but is not implemented for the Apple Operating System (This latter operating system has its own graphics protocol, MacWorkstation, which we are experimenting with).
- **Common Window Application standard/implementation**: We have not been able to give adequate attention to this item because of staff and budgetary constraints. It is worth noting that the Common Lisp User Environment (CLUE) is a window system defined on top of CLX and is currently in use in the KSL.
- **Develop/Extend Virtual Graphics applications**: Very little progress has been made in this area because of staff and budgetary constraints. We intend to emphasize the development of virtual graphics applications beginning this summer.

3.3 - Distributed Mail System

- **InterLisp mail reader/composer**: This software is completed, and is now part of the Lyric TCP-full-sysout. No further Xerox work is planned.
- **Redesign IMAP protocol (IMAP-2)**: This has been completed.
- **2060 IMAP-2 Server**: This has been completed.
- **UNIX IMAP-2 Server**: This has been completed, and is currently being alpha tested.
- **Common Lisp mail reader/composer**: The TI version should be completed and in initial testing by the end of June 1988. No work is planned on a Symbolics version. There are similarly no plans to translate the Xerox InterLisp client into Xerox Common Lisp given Xerox's plans to move to SUN's, and the existence of InterLisp within the Xerox environment for the foreseeable future.

- **Update Common Lisp IMAP-2 Clients**: This is completed.

- **UNIX mail client/reader/composer**: This project is on the backburner because of staff limitations, and the small number of SUN clients in use at SUMEX-AIM (a Macintosh-II client is underway for the Apple operating system rather than AUX, since the former is the primary OS in use at SUMEX-AIM). We have imported a UNIX version of the 2060 mail reader/composer called MM-C which was written at Columbia University. It is not a distributed mail system in that the reader/composer and mail file are assumed to reside on the same machine. The MM-C system will be used to provide national community and home mail services on the SUN-4 until further versions of the IMAP-2 clients are available.

- **Enhance reader/composer tools**: The reader/composer tools have undergone significant continual development since their release to our local user community. For example: message filtering was introduced earlier this year and one can now filter on free text searches; subject, from, to, cc and bcc text searches; new recent and old messages; On, before or since a given date; messages that are Seen/Unseen, Flagged/Unflagged, Answered/Unanswered, Deleted/Undeleted; and message keyword searches.

3.4 - General Computing Environment

- **Lisp and AI shell environments**: Common Lisp now runs on Xerox systems using the Lyric sysout, and Lucid Common Lisp is on our SUN workstations. The Macintosh-II supports Coral Common Lisp. Finally, with the advent of the microExplorer co-processor on the Macintosh II, the entire Explorer Common Lisp environment is available on Macintosh II's configured with this board and an Ethernet interface.

  In addition, we have made considerable progress in analyzing the performance of Lisp systems on various kinds of hardware, with an eye toward guiding our work on future Lisp systems and the trade-off between specially microprogrammed Lisp machines and implementations on standard workstations. We have also defined the requirements for a powerful Lisp programming environment based on the key features of the Xerox, Symbolics, and TI environments that we use routinely in our AI research work.

- **Distributed File Support**: The Xerox Common Lisp RPC/NFS implementation has been completed, and Xerox has a strong interest in acquiring this software from us and including and supporting an enhanced version as part of their standard system release. We would in turn receive all improvements to the code. Both TI and Symbolics have released an RPC/NFS implementation. Because of budgetary and staff limitations this year, we have be unable to make any progress on *Network management, backup, and archiving tools.* This area has been given a high priority and we will begin to work on it this summer.

- **Distributed information access**: We have installed SUN UNIFY (a powerful relational data base system) on our SUN file server and implemented a Common Lisp remote procedure call/SQL query interface for Lisp machines to experiment with remote data base access. We have also been experimenting with the Apple HyperCard system for organizing and disseminating information in a distributed community.

- **Operations management tools**: We have made little progress in this area during this report period.

3.5 - Phasing of the transition to a distributed AIM community

Experiments in the Stanford/AIM community: Each new piece of hardware or software has been tested initially by a selected subset of the Stanford/AIM community, e.g., members of the systems staff who are willing to put up with problematic software in the alpha test phase. It has then been beta-tested by a larger subset of the same community, and then released to any interested member of the community as a whole. A typical example is the Xerox IMAP Client, MM-D. After several months of alpha testing by two or three members of our systems staff, the software was distributed to other KSL research staff members for beta testing and suggestions for improvement. Finally, it became a part of the standard Xerox Lyric system used by the Stanford Knowledge Systems Laboratory.

4 - Remote Workstation Access, Virtual Graphics, and Windows

4.1 - Remote Access

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.

4.2 - Virtual Graphics

In order to link the output of workstation displays across networks, 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. A mechanism for doing this is called a remote graphics protocol.

As reported last year, we selected the MIT Project Athena X window system [4] as the remote graphics protocol standard for our work, and noted that X is a very complete protocol that has been developed over the past several years at MIT[1]. We also reported that an X *client*[2] for Texas Instruments Explorers was being written here at SUMEX-AIM, and that TI in conjunction with MIT was developing a *server*

---

[1]The X protocol was completely redefined last 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.

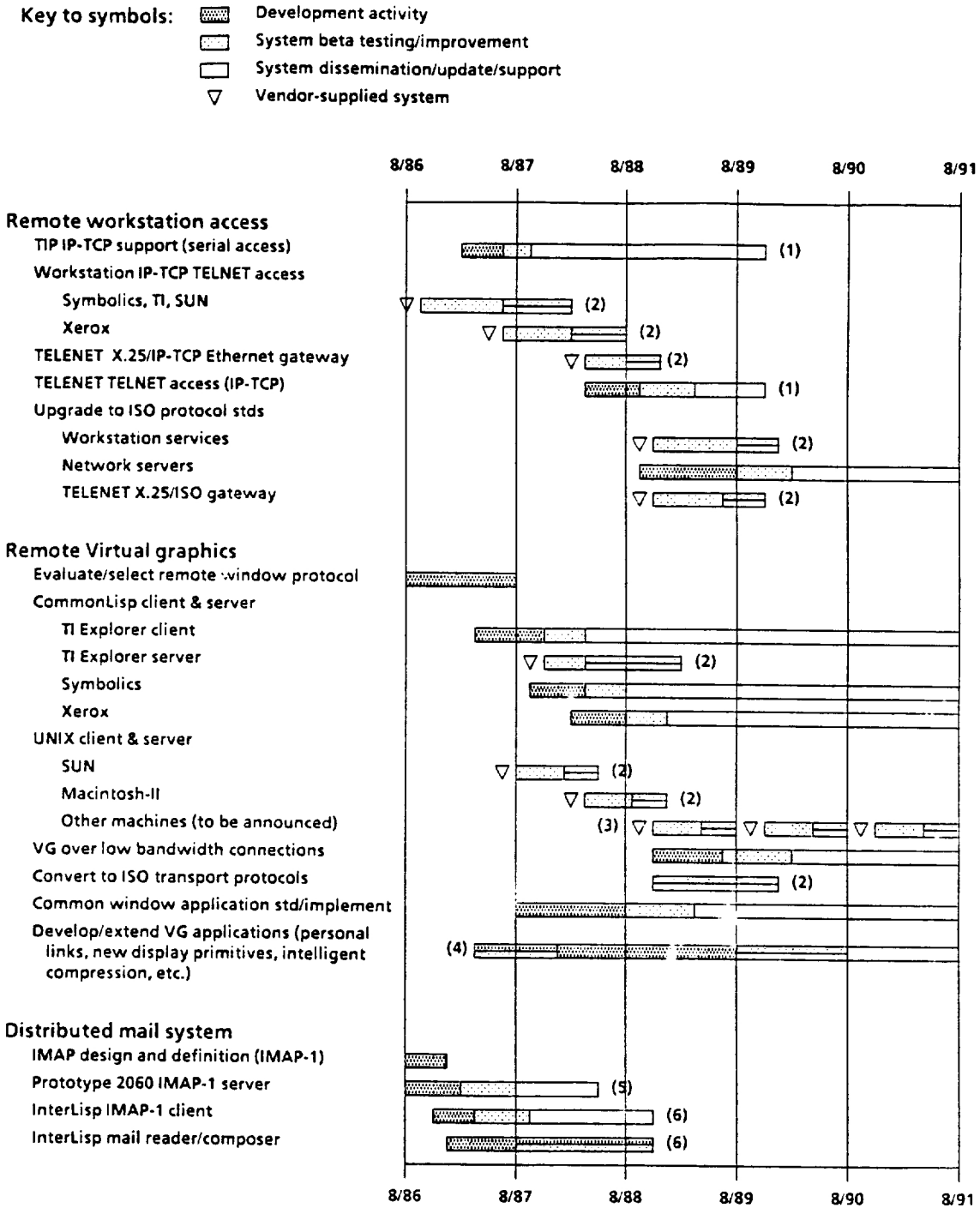

Figure 1: Core System Development Schedule

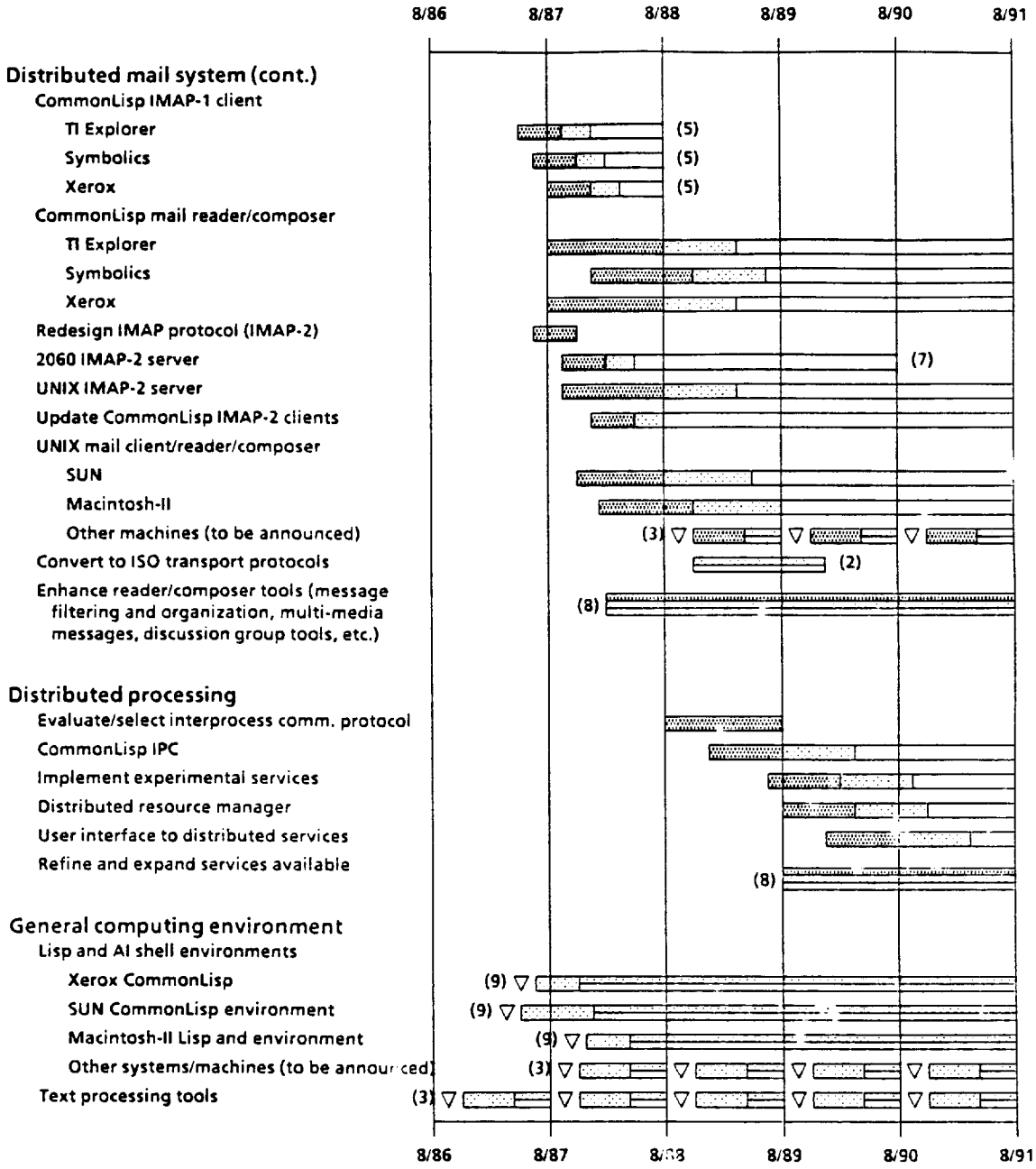

Figure 1: Core System Development Schedule, Continued