Artificial Intelligence—Definition and Practice

ASA B. SIMMONS AND STEVEN G. CHAPPELL

(Invited Paper)

Abstract—**A new class of computer systems has emerged which makes extensive use of the fact that computers operate equally well in processing either numbers or symbols. The best known of the systems which exploit this capability are the expert systems coming into wide use in industry. The research from which these systems are derived is called artificial intelligence. In spite of the popularity of systems based on this technology, there is still confusion about the meaning of the terms, and how the technology can be effectively used. This article presents an overview of the field, with definitions, recommendations, and an example. A definition of terms is presented, followed by a brief history. Potential roles which a symbolic processing system might play are discussed, with a brief summary of those which have been most successfully used. The various methodologies in use and their respective strengths are discussed. Important factors to be considered in selecting a problem are mentioned, followed by a discussion of the hardware and software tools available. The discussion of tools includes an evaluation of the kinds of systems for which they seem to be best suited. The implementation effort is discussed in detail, because it has often been underestimated. Commitment of key personnel is a recurrent problem, and the evocation of key elements of the knowledge used represents an especially difficult problem. Therefore, the issues of knowledge acquisition and knowledge representation are discussed in some detail. Current arguments about the best language to use, such as the choice between Lisp and Prolog, are addressed. A practical example is described in some detail, to illustrate how the various elements of the technology may be blended to achieve a useful autonomous underwater vehicle. The paper closes with a discussion of current research issues, including the types of machines being investigated. The strengths and potential of parallel architectures and neural net machines are briefly discussed, because of their potential impact on the field of all processing.**

Keywords: **artificial intelligence, knowledge engineering, expert systems.**

I. Purpose

A NEW CLASS of systems has emerged which exploits the fact that computers actually cannot differentiate between representations of numbers and representations of symbols, and therefore have the capability to do symbol processing as easily as they do number processing. In its most common form, this type of system is called an expert system (ES). The ES originated in laboratories conducting research into ways in which digital machines might be made to mimic intelligent human behavior. The nature of the research involved caused the term *artificial intelligence* (AI) to be applied to it and to all of the technology developed from it. In spite of the wide use of the resultant technology, it is not always clear what the specific meaning of the term is, or how problems can be identified as candidates for application of the methodology. The intent of this overview is to provide some definitions, plus some insight into the origins and usages of the most common methodologies developed as a result of AI research.

Manuscript received October 30, 1987; revised February 9, 1988.

A. B. Simmons is with the Submarine Signal Division, Raytheon Company, Portsmouth, RI 02871.

S. G. Chappell is with the Marine Systems Engineering Laboratory, University of New Hampshire, Durham, NH 03824.

IEEE Log Number 8821754.

The current proliferation of ES's results from the simultaneous occurrence of several events. These include an awareness of the power and usefulness of symbolic processing. They also include increases in the speed of processors. But perhaps most important were drastic reductions in the cost of both processing power and memory. Increased capacity in both of these elements has made the use of sophisticated graphics and a more natural language interface practical. This has been accompanied by a reduction in cost which makes the use of personal workstations reasonable. Also, the availability of inexpensive expert system shells has allowed many people to try their hand at developing expert systems (rule-based, mostly) on their personal computers. The capability now available in low-cost workstations has therefore accelerated the spread of tools from the AI laboratory to industry and this trend is expected to continue.

A definition of the term *artificial intelligence* is followed by a brief summary of the history of the field, including some of the projects which defined and demonstrated the methodologies. Guidelines are presented which have proven useful in determining the utility of AI techniques in solving particular problems. These guidelines describe the general characteristics of a problem which make it a good candidate for applying AI to its solution.

It is intended that the potential user of AI get sufficient information from this review not only to identify the utility of the AI approach to his specific problems, but also to be able to define reasonable limits for his expectations. It must be emphasized at the beginning that the tools of AI provide a means by which human expertise may be captured in a machine, thus allowing it to solve problems previously solved only by the human. They do not allow solutions of problems which have never been solved before, or for which the solution procedures are not implied by successful human behavior. This is the foundation for the proposed definition of artificial intelligence:

> The term *artificial intelligence* denotes behavior of a machine which, if a human behaves in the same way, is considered intelligent.

It is difficult to extend this definition, because the definition of what factors describe human intelligence is not clear.

One alternate definition does exist which captures the nature of the work being done in the field. This was proposed in Rich [81], and reads as follows:

> Artificial Intelligence is the study of making computers do things which, at the moment, humans do better.

The second definition has the advantage of capturing the current preoccupation with clever use of von Neumann machines (given the usual image evoked by the term *computer*), rather than investigation of new kinds of machine, but it ignores those researchers who are working on the realization of fundamentally different machines, such as the connectionist, or "neural net," machine. Even though neural net machines are designed and built by people outside the current group of AI researchers, they are so closely related that they must be included. Therefore, the Rich definition is now too restrictive.

These definitions contrast sharply with the goal stated in Haugeland [57]:

The fundamental goal of this research is not merely to mimic intelligence or produce some clever fate. Not at all. AI wants only the genuine article: *machines with minds*, in the full and literal sense.

Very few workers in the field would agree with this statement, and most seem to feel that it is important that expectations be kept to a reasonably low level. It will become clear, in the following discussion, why the goal stated by Haugeland is not applicable to systems now being developed. Even so, Haugeland has presented an excellent history of the field and his book makes a good supplement to this discussion.

The confusion about AI may stem from the fact that, in humans, intelligence seems to involve both an overt ability to solve a specific class of problems and a latent ability to discover solutions for a new class of problems. These two kinds of capabilities are quite distinct. Confounding them leads to expectations that an ES will provide new solutions, when it cannot. For example, current systems do not have the ability to learn, in any meaningful sense, and they do not have the ability to do induction, in the sense of creating new concepts by abstracting from experience or by rearranging pieces of knowledge in novel ways. Above all, that is, they do not experiment.

AI technology does provide a set of tools which allow some aspects of human behavior to be easily transferred to a machine, and the techniques used encourage a new kind of thought about the nature of such behaviors, because they focus attention on the *type* of knowledge involved, as well as a plausible representation of it. They provide a framework for implementation of a known, but inexact, method of solving a problem. They do not provide the solution itself. The capability provided therefore resembles most closely a new form of calculus, which may or may not be applicable to the problem at hand. It is still necessary for the user to choose a specific technique, based on his understanding of the behavior involved and the structure of the problem. The user must supply all the information, knowledge structures, and operations needed. It is therefore better to refer to the new type of endeavor as knowledge engineering, and the systems created as knowledge-based systems. The resulting system will always be limited to the subset of human knowledge embedded in it, although its behavior may at times be unexpected. The scope and adequacy of the implemented system will, in any case, be limited by the insights of the people whose knowledge is being captured, and by the skill of the knowledge engineer who is doing the capture and transforming it to machine form. The system will also be limited by the extent to which the selected tools represent a "good fit" to the components of the problem and its solution.

The discussion is divided into six parts. These are

- the brief history mentioned above,
- a definition of common terms,
- a discussion of possible uses: why and when,
- an overview of available tools,
- a discussion of current implementation issues,
- a description of a specific application in the field of ocean engineering,
- a discussion of research issues.

An extensive reading list is provided. This list covers not only the references cited directly in the paper, but also articles and books containing discussions of specific developments and some of the philosophical issues involved. For example, in the discussion some previous projects will be mentioned which have been instrumental in developing and demonstrating the various tools now in use. These will not constitute a complete summary of the work in the field of AI. However, some of the most notable are described in Waterman [86]. The list of references appended has therefore been made entensive enough for detailed examination of the background and original use for all of the topics and tools discussed, plus others. It also allows study of some of the philosophical issues which have arisen as the use of the tools has become more extensive.

References [1] through [25] provide insights into the origins of many ideas developed in studies sponsored by government agencies which have affected implementations. The Berliner papers [2]–[4], [29]–[34] are important because they describe the evolution of the design of the current machine chess champion, a machine which has performed quite well against human grand masters. This design is an interesting use of parallel processors, and uses a unique variation on the "min/max" search strategy for decision trees. References [1], [5], [6], [10], [11], [13], [15], [18], [20], and [25] explore issues in the structures of knowledge representation and control mechanisms. Reference [7] describes goals which were met in a distributed design used in a system developed at MITRE, which turned out very well (see Waterman [86]). References [8], [12], [14], [17], [22]–[24], and [67] discuss potential roles that knowledge-based systems might play. References [9] and [21] describe some of the problems involved in identifying the nature of the human competence being captured. Reference [21], in particular, by identifying the unique capabilities of vision and hearing, illustrates the difficulty of providing a single mechanism which mimics human capability with both senses in a single design. Reference [1] represents an interesting attempt to completely formalize human reasoning behavior, but is extremely turgid. References [27] and [45] illustrate some of the thinking about intelligent machines reflected in science fiction. The publications by Borning [35], [36] are extremely important, because they show how simply and effectively all of the systems created by humans, both concrete and abstract, can be designed, modeled, and tested in a simple object-oriented environment (in this case, Smalltalk

'77). References [37]–[39], [50], [52], [71], [73], [77], [78], [82], [85], [90], and [92] offer additional views of some of the issues involved and the approaches taken in addressing them. The problems of explaining behavior by inferring the hidden processes involved in cognition and learning are illustrated in [28], [40], [44], [46]–[48], [52], [53], [56], [58], [63], [66], [77], [81], and [88]. The background of Lisp is covered in [69], and the use of Lisp is covered in [85] and [94]. The original impetus for using both Boolean logic and the binary form of perceptron is given in McCulloch and Pitts [70], soon to be republished by MIT Press in a book of selected papers being edited by Anderson and Rosenberg. The issues involved in designing a language for AI research, and in programming in general, are covered in [43], [57], [62], [64], [84], and [85]. Smalltalk '80 [54], [55], [65] is especially important, because the object-oriented paradigm is becoming more and more pervasive in knowledge-based system design. The discussion of fuzzy logic in Negoita [74] is important, because it represents the first attempt to formalize the human capacity to work with flexible measures and with objects which may belong partly to one class and partly to another. The ease with which natural language can be used when one is willing to include knowledge of semantic functions is made clear in Winograd [90], the source of the "blocks world" so often cited in the literature. The issue of the Japanese lead in the field of computers based on the insights of AI is discussed in Feigenbaum [49]. References [68], [72], [80], and [89] discuss neural net machines, the brain operations on which they are based, and some of the design issues. Minsky and Papert [72] is important because the negative evaluation contained therein halted funding for neural net machine research for about 20 years. Lippman [68] is a good, short survey of the field of current machines and dominant implementations.

To illustrate the practicality of AI technology and methods, a detailed example is given. The example chosen shows how the various elements of the technology may be artfully blended to achieve a system capable of reasonable behavior. The system selected is an unmanned, untethered (i.e., fully autonomous) underwater vehicle. This design is especially interesting because it makes no attempt to reflect expertise, per se. Instead it focuses on two facets of the behavior spectrum:

1) real-time reaction versus reflection before action;
2) "reasonable" behavior versus "expert" behavior.

The system is structured so that expertise can develop gradually, and can be added to the system without requiring an extensive redesign. Because of this, it represents a uniquely attractive use of all methodologies, and it is an especially good example of the utility of the techniques of using multiple blackboards and using the object-oriented paradigm.

II. History

The research which led to the current AI technology started after the Second World War, and was spurred in large part by the advent of the digital computer. However, the foundations were laid much earlier. The "computing engine" of Babbage provoked much discussion on the potential of machines, in the role of robots, to evince intelligent behavior. Later, George Boole developed the Boolean algebra as a means of describing possible operations of the brain in a formal calculus. The early period culminated with the work of Alan Turing, who enunciated the Turing test, and who described the design of the Turing machine. Turing's work provided the standard against which all machines and grammars (including human grammars) were evaluated for many years.

Support for the ideas of Boole was provided in the late 1940's by McCulloch and Pitts [70], who discovered that some brain cells operate in the binary fashion required by Boolean algebra. Since the same kind of logic was being used in digital computers, there was an implication that the computer should now begin to reflect more aspects of human behavior, which led to more discussion.

In the same time frame, von Neumann defined the machine architecture which bears his name and spurred the use of digital computers as general-purpose devices. The fact that programs and data could be stored in the same memory meant also that the machine could work on different tasks by simply installing the required set of instructions and data, with a mechanism to select the appropriate set when needed. This led to the hope that a single type of machine could adequately solve all problems, which resulted in a focus on finding ways to make it behave intelligently. The preoccupation with clever operations in the von Neumann machine introduced a bias which is still evident in AI research.

In the mid-1950's, research started in earnest on ways of mimicking human behavior by processing images (or symbols), primarily at MIT and Stanford. The research originally followed two paths: development of unique hardware and development of programming tools for von Neumann machines.

Hardware research culminated in the perceptron, which was so severely criticized in Minsky and Papert [72] that research on what are now called neural net machines stopped for some 20 years.

The term *artificial intelligence* was coined in this time frame to describe the research being done.

A belief in the efficacy of stepping through simple rules, combined with a belief that the operations of the Turing machine accurately model human behavior, led to claims for behavior achievable in digital computers which could not be substantiated. The turning point in this phase seems to have been the project for automatic translation of languages. This failed so badly that it provoked a complete cessation of funding for AI research in Great Britain and a severe reduction in the United States.

The unsatisfactory outcome of projects undertaken in the 1950's caused the whole field of AI research to become relatively dormant until the mid-1960's. Soon after its renaissance, impressive machine behavior was achieved, especially in problem solving and diagnostic assistance. The ability to reflect the behavior of an expert was impressive, even though the range of behavior was quite limited. Landmarks of this period include:

- development by McCarthy [69] of the LIst Processing

(Lisp) language, which focuses on recursive processing of arbitrary lists;

- the "blocks world," from *Understanding Natural Language,* by Terry Winograd [91], which demonstrates limited communication with a robot in natural English and with automated planning;
- The Eliza program, which mimics a Rogerian psychotherapist, by Weizenbaum [87];
- *The Semantic Organization of Memory*, by Quillian [73], which outlines a system of world linkages reflecting concept linkages in natural language;
- the MYCIN program [42], which demonstrated the utility of rule sets and confidence factors in developing a diagnostic adviser for physicians.

Projects such as these supported the view that existing machines, given the right operating system and the right set of rules, could accurately emulate important kinds of human behavior, especially when the role of the machine was carefully selected. They also supported the view that Lisp was the language of choice for investigating intelligence, because they were done in one or the other of two dialects of Lisp: the Interlisp dialect (BBN, Xerox, and Stanford) or the MacLisp dialect (MIT). The Interlisp-D dialect dominated in this work, because it was the dialect used at the Palo Alto Research Center of Xerox (Xerox PARC), and because it reflected an emphasis on user interaction resulting from the work of researchers such as Alan Kay. The importance of the Xerox PARC work of this period can be judged by the fact that the Macintosh computer, the Sun workstation, and the Apollo workstation, as well as all comparable machines, use ideas generated during this time at Xerox PARC. However, the field remained relatively dormant.

These developments, coupled with the split of software activity in academe into the separate discipline of computer science, caused a divergence of research activities on robots and research on machine intelligence. Researchers in machine intelligence (AI) have not been especially concerned with robotics, and roboticists have not generally been concerned with (or used) AI. Researchers in AI have been primarily concerned with programming existing sequential machines, accepting as given the constraints thereby incurred, because software has been their primary concern. Roboticists, on the other hand, have been primarily concerned with hardware useful at the end points of behavior sequence: sensing and motion. Sight, touch, "hearing," and motion control have been treated by roboticists as amenable to deterministic approaches (i.e., not involving heuristics). For this reason, the subject of robotics is not covered in this review.

The beginning of the 1980's saw a proliferation of projects based on the work done in the late 1960's. During this time, a new organization became prominent: Carnegie Mellon University. This group developed a rule-based problem solving system called R1/XCON. This was developed under the auspices of DEC to provide assistance to people responsible for translating purchase orders into collections of specific units and cabinets, including cabling orders. The system developed for this purpose was one of the first to be labeled an expert system, because it was based on the behavior of especially talented people. The architecture used, an "inference engine" plus a "knowledge base," became the prototype of the class of systems now denoted by the term *expert system.* The success of this work gave impetus to the application of AI technology of all types, but especially inference engines, to all kinds of practical problems.

Several companies appeared on the market with practical products. Hardware for machines specializing in the Lisp language ("Lisp systems") was presented by Symbolics, Lisp Machines, Inc. (LMI, now Gigamos Systems), and Xerox. The OPS5 software system was made available for the VAX computers, running under Unix. Comprehensive, specialized software for knowledge engineering on Lisp systems was presented by Intelligenetics (now Intellicorp), Inference Corp., and the Carnegie Group, among others. An excellent tool for use on the Macintosh (Nexpert) was presented by Neuron Data, Inc. The field had now matured.

Two types of methodology have become dominant. The first is concerned only with sets of rules and their activation. The second is concerned with more general problems of the organization of knowledge, plus a need for cooperation between various sources of knowledge in a synergistic way. Systems using only the rule-based approach focus on capture of appropriate rules, the facts used, and proper rule execution in a limited, specific situation. These are the kinds of systems properly referred to as expert systems, since they focus on capturing the insights of an expert in a small domain, and coding these insights. The second type of system encompasses a larger domain than just facts and rules for one situation. The latter type of system is better referred to as a knowledge-based system.

While expert systems have been dominant, it is also realized that they have limitations. These limitations are being overcome in the more generally oriented knowledge engineering systems by the adoption of other methodologies. These include "state-transition networks" and "case grammars." The "frame" construct, a form of knowledge organization proposed in Minsky [1], has, in particular, become very important in knowledge engineering. Thus, several very powerful systems for organizing and using knowledge were beginning to evolve, based on a combination of all of the insights of the late 1960's and early 1970's.

The major limitation of rule-based systems stems from difficulties encountered in extending or modifying rule sets. This is especially true when the only form allowed is the "conjunctive normal" form (only combinations containing "and") as is recommended in using Prolog. When the rule set become relatively large, there are problems with tracing the effect on other rules resulting from changes in a specific rule, and with predicting the effect on existing rules resulting from the addition of new rules. This stems from two facts:

1) There will implicit "or" operations because several rules produce the same result.
2) There tend to be some rules which trigger activity when it is inappropriate.

It is now recognized that one of the primary abilities of a human is selection from a large set of rules of a small subset to apply in a given situation. Recognition of the situation is crucial in this process. It is not clear how the same kind of recognition can be easily done in a system which is built of only rules and an executive which activates a single rule for each step. The use of the frame methodology alleviates this problem.

The final result for the field of AI is the emergence of products and applications, discussed in the remainder of the paper, most of which involve use of the ES approach.

III. Roles and Intelligent Activity

The definition of intelligent activity is as described above: observing behavior in a machine which evokes the term *intelligent* when the same behavior is observed in humans. Granted that this type of behavior can be evoked from machines, the question becomes the extent to which the machine can then substitute for the human in the same type of situation.

There are several roles the machine might play:

- ''gofer'': getting routine data, or performing simple, repetitive tasks;
- intelligent assistant or adviser, who can help keep track of information and progress, providing advice when needed;
- replacement for the human in a specific type of activity.

The analysis activity involved in all of these roles includes not only awareness of relevant facts and their implications, but also the recognition process. The recognition process, in turn, requires retrieval of a model which defines not only all of the relevant facts, but also all activity, assumed behavior, and evaluation procedures needed to plan and execute a program of activity.

The role of human replacement is often thought of in terms of the speculations in science fiction stories, where the machine replacement is ''smarter'' or ''better'' than the human replaced (or, as hypothesized in Delaney [45], a human reprogrammed in a machine language to act like a machine). This possibility is not seriously considered. Replacement of humans *is* considered, but in a very restricted sense, in situations where the environment is very hazardous. However, there is no expectation of intelligence beyond the hope of asymptotically approaching human capability, and none of the successful systems now in place are of this type.

The ''gofer'' type of system can be implemented on existing machines if the range of activity is severely constrained. If implemented on a Lisp machine, an interesting level of sophistication can be achieved, as shown in the blocks world robot of Winograd. That is, the system can have an ability to absorb new information, to answer questions, and to carry out orders, all its inputs being in normal English, and all of its responses also being in normal English. The key point is that the universe of activity and meaning must be severely restricted, as indicated by Winograd. Within this constraint, the machine can provide excellent support for users. This has led to a proliferation of machines operating as an intelligent assistant, adviser, or tutor.

The major effort, therefore, is currently focused on development of systems which can use the knowledge of existing experts to develop machines which can increase human capability by acting as ''knowledge amplifiers.'' Two goals seem to be common. One goal is the use of heuristics to do low-level processing, thus speeding up a task while freeing the human to do more creative work. The other goal is to improve the performance of individuals by making available the advice and guidance of an expert. Success in achieving both of these goals is a major reason for the popularity of the ES approach.

The prevalent implementation procedure requires involvement of several types of people to realize the desired system. It involves the activity of the hardware experts and programmers traditionally involved in developing digital systems. It also involves a direct interface between a knowledge engineer, who specializes in discovering and encoding knowledge, and an expert. The tools and the hardware to be used will generally have already been selected at the time the implementation starts, so that *a priori* constraints are imposed by the design of the hardware and software selected. Because of these *a priori* constraints, care must be taken to avoid a ''black hole'' situation, in which increasing effort results in decreasing effectiveness.

A lot of human interaction is required. The need for a new type of engineer—the knowledge engineer—results partly from the fact that experts often cannot explain precisely why they make specific decisions, so that simple interviews are an inadequate way of capturing the relevant cognitive process. More often, a careful study of context and activity, using the skills of an objective observer, is required to discover the most likely triggers of relevant activity. The problem is quite similar to the discovery process in which anthropologists, linguists, experimental psychologists, and cognitive psychologists (among others) are trained. This is why it is often found that people from these disciplines make better knowledge engineers than people trained in engineering or computer science.

It is recognized that this situation is unsatisfactory, as it adds a new layer of personnel to the layers that already exist, the number for standard systems being shown in Fig. 1(a) and the number for knowledge-based systems being shown in Fig. 1(b). Currently, the usefulness of a machine created independently by hardware engineers is obtained as a result of the activities of a programmer, who translates agreed specifications into machine behavior using a high-level language overlaid in the machine language. The emergence of the knowledge-based system created a need for the specialist in both discovery and programming to translate observed behavior into the acceptable behavior of an existing machine. The tools of the knowledge engineer are dialogue and observation in ''representative'' situations, followed by extraction of sequences and situations which seem relevant. This is an inexact procedure, which gives varying results. The uncertainties involved in articulating ''logical'' sequences for behavior which is not, in itself, clearly sequential or logical creates further problems, especially when an attempt is made to merge the rationalizations obtained by working with several experts. Therefore, the eventual goal is to let the expert have a machine with which he communicates directly, and which learns directly from him, as shown in Fig. 1(c). In this final phase,

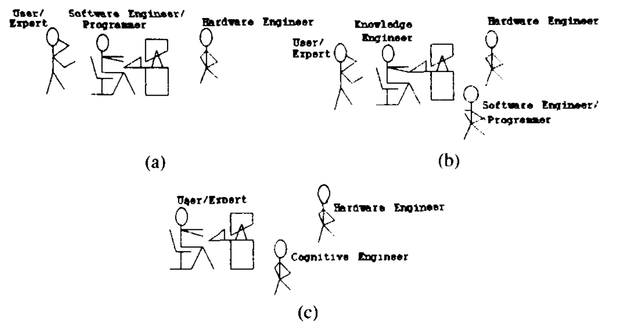


Fig. 1. Trend in capability of computer systems. (a) Standard computer system. (b) Knowledge-based system. (c) Cognitive system (future).

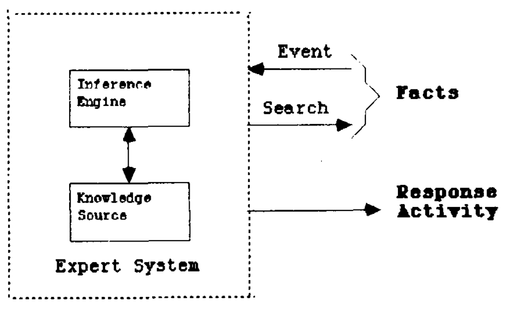


Fig. 2. Expert system.

the user works alone with a self-organizing machine, teaching it directly to behave as desired. This phase requires the development of machines which depart radically from current von Neumann designs, and is most likely to arrive when the neural net machine technology matures and fuses with other technologies.

IV. Definition of Terms

The most common terms for the basic structures of the knowledge-engineering discipline are

1) expert system
2) semantic network
3) frames
4) blackboard.

The single ES is envisioned as consisting of two parts: an inference engine and a knowledge source, as shown in Fig. 2. The inputs to the engine are events. These may be the statement of a new fact or the negation of a known fact or belief. These events may be spontaneous, as in the case of a sensor, or may result from a search for possibilities, as in the case of medical diagnosis. The knowledge source consists of those rules needed to react properly to any combination of values of the facts within the purview of the system. When a new fact is discovered, or when an event necessitating assertion of a new fact occurs, the inference engine examines all of the relevant rules to determine whether or not a response is appropriate. These rules all have one of two forms: "if A, then B," or "C is implied by A and B." In the form "if A then B" the first part (left-hand side), is called the premise, and the second part is the "production." In the form "C is implied by A and B," the first part is the fact to be asserted wherever both A and B are true. The second form reflects an approach based on symbolic logic.

The production specified by any rule of the first form is executed when the rule is activated by a specific combination of true and false values of facts. In the event that the premises of more than one rule are satisfied simultaneously, a "conflict resolution" procedure is invoked to select one production only.

The meaning of the second form for a rule is often taken to be "A and B must be done before C occurs."

The definitions and operations used in both kinds of rule make this kind of system realizable in symbolic logic because of their declarative nature, and this has spurred the development and use of the Prolog language for rule-based systems.

The semantic network structure has been most commonly used in defining taxonomies. Typically, it is shown with two kinds of link: "is-a" and "a-kind-of," as shown in Fig. 3.

In this structure, each higher level node is considered to be the parent of all of the nodes below. Each lower level node is the child of all of the higher nodes to which it is connected. As a result of this linkage, each child node can be made to automatically "inherit" all of the attributes standard for its class, with a provision to override the standard inheritance with unique attributes, if desired. In addition, each node can have unique attributes applicable only to itself and its children.

The instance shown is a very simple case. The links can be bidirectional, with the added label "is-part-of," if desired. It is also possible to have other types of label, such as "has-a" or "can," and to have as many types in a single network as desired. Each node can belong to as many sets as desired, and can thus "inherit" characteristics from as many "parents" as desired. These links are a natural extension of the "attribute list" construct in Lisp. The number and types are limited only by the number of pointers which can be stored. This type of construction is used in the search trees (see Charniak [42], Rich [77], or Winston [92]). It also serves as the basis of object-oriented programming (see Stefik and Bobrow [84]), where each of the nodes represents an object with knowledge and the ability to process messages. There is a problem in applying these concepts to natural language, because every node contains many other nodes within itself when defined as a part of the language (see Quillian, in Minsky [73]).

The most common use for the search tree format is in representing the activity required to achieve a specific goal, given a set of facts. This occurs in diagnostics, identification, and games. In these cases, it can be assumed that all of the facts are independent and can be linked together in all possible combinations. Achievement of a desired goal then involves tracing the paths through the tree in the most efficient and economical manner. The two simplest approaches involve the use of "breadth-first" or "depth-first" searching. In the case of breadth-first, all of the nodes at the current level (called a ply) are examined before stepping to the next level (this corresponds to the kind of operation followed by a simple rule-based system such as OPS5). In the case of depth-first, a single path through all of the nodes succeeding the current node is traced to the "bottom" of the tree, before path branching from

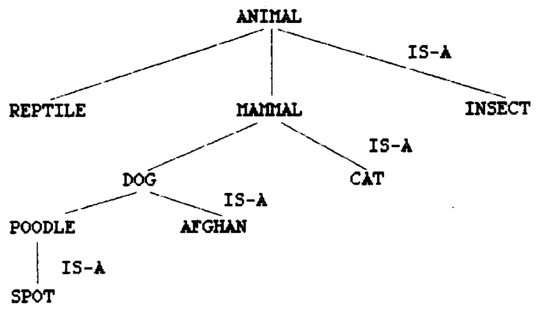
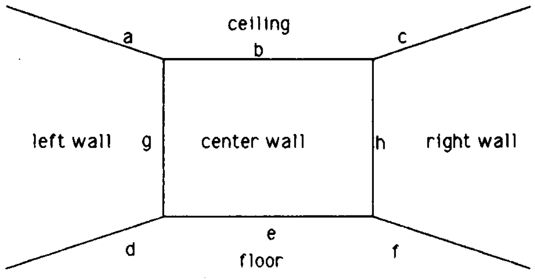


Fig. 3. A basic semantic net.

the right-hand neighbor is traced. In either case, the search normally stops when one of the terminal "leaves" is the desired goal. The complexity of the operations involved is typified by the problem of playing chess. Exhaustive search of all possible combinations of plays is impractical, even if a supercomputer is used. The basic problem with this approach is that the search must operate in an informed way; if the problem is combinatorial, explosion is to be avoided. In the METADENDRAL system for identifying organic compounds (see Waterman [86]), this was done by using the "cause-effect" constraints imposed by knowledge available from additional sources to constrain acceptable combinations of nodes and branches in the tree, drastically pruning it, and thus drastically limiting the size of the search space. In the case of chess, the tree must be pruned by a combination of search through several layers and assignment of values to paths which include moves by the opponent. The most successful example of this is the system of Berliner [29]–[34], which uses a system of cooperating experts coordinated by a strategy manager. The ratings applied by Berliner to dynamically choose the best path, thus dynamically pruning the tree, follow the same philosophy as the "confidence-factor" approach used in systems for diagnostics (see the MYCIN system (Davis and Lenat [44])). The use of causal reasoning to constrain the search is covered very nicely in Davis [25]. The element common to all of these approaches is to direct the search along promising branches in the paths through the tree. The general name for this kind of operation, where only a few candidate branches are selected for further examination, is beam-directed search. This type of search produces a more efficient and satisfying result than the simple stepping procedures used in either the depth-first or the breadth-first operations.

The other major construct in common use is the "frame." Fig. 4 shows the sample of this construct as defined by Minsky. In this case the frame will attempt to capture the features characterizing a "dining room," one example of which is given in Fig. 5. The data structure representing the significant elements is shown in Fig. 6. Notice that frames can be nested within frames. This construct was defined to describe a metalevel construct which might help to account for some human behavior in handling complex situations and scenes. It attempts to incorporate the use of partial knowledge by postulating the existence of "defaults," where the default is some concatenation of fundamental characteristics (called significant features) which allow a wide range of objects to be



(a)

Frame Item:	Defining Characteristics:
Wall:	
Left :	Edges "a", "d" & "g"; Corners "ag" & "gd"; etc.; no inconsistent combinations
Center:	Edges e, b, g & h, Corners "egd", "efh", "bag" & "bch", etc.; no inconsistent combinations
• • •	
Contents: Table: Chairs: etc.	Defined by Dining Room Frame

(b)

Fig. 4. Basic frame concept. (a) Frame elements, room. (b) Dining room—basic expectations to which defaults are assigned.

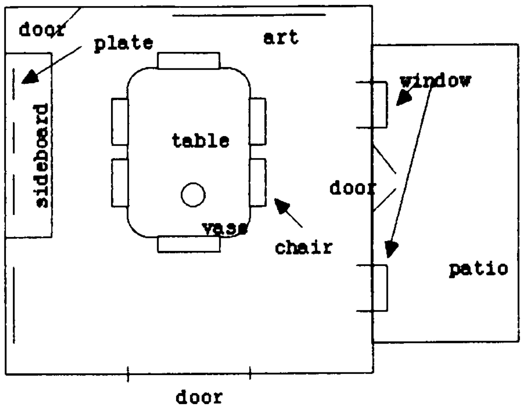
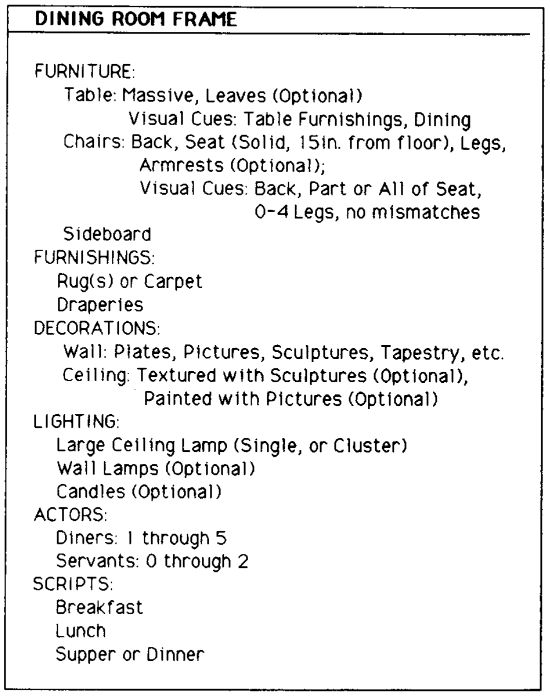


Fig. 5. A dining room frame.

plausible. It also attempts to account for the invariance of perception in the presence of translation and rotation. In terms of reaction to a situation, it postulates a selectivity operation based on the perception which allows humans to bring to bear a small subset of knowledge and rules appropriate to the task at hand. In this view, there exists some "typical" case which somehow constrains the processing and interpretation of facts to that set which matches expectations.

The data structure defining a frame consists of a list of attributes (or pointers to sets of attributes), procedures, and rules which describe the "typical" case of the object or situation being analyzed. In principle, all the attributes listed are unique to the object, and constitute the necessary and



```
DINING ROOM FRAME

FURNITURE:
   Table: Massive, Leaves (Optional)
           Visual Cues: Table Furnishings, Dining
   Chairs: Back, Seat (Solid, 15in. from floor), Legs,
           Armrests (Optional);
           Visual Cues: Back, Part or All of Seat,
                        0-4 Legs, no mismatches
   Sideboard
FURNISHINGS:
   Rug(s) or Carpet
   Draperies
DECORATIONS:
   Wall: Plates, Pictures, Sculptures, Tapestry, etc.
   Ceiling: Textured with Sculptures (Optional),
            Painted with Pictures (Optional)
LIGHTING:
   Large Ceiling Lamp (Single, or Cluster)
   Wall Lamps (Optional)
   Candles (Optional)
ACTORS:
   Diners: 1 through 5
   Servants: 0 through 2
SCRIPTS:
   Breakfast
   Lunch
   Supper or Dinner
```

Fig. 6. Dining room data frame.

sufficient set to characterize all members of the set (as in "dining room" and "dining room table" in the example). In addition, the attribute list contains some characterization of plausible boundary conditions, such as the character of connecting objects (one dining room door normally opens onto a kitchen, not a beach, as in the Minsky example). That is, the frame accurately characterizes a prototype, so that any item of the generic type is instantly recognized, no matter what its variations, within some kind of intuitive limits. However, there are severe problems in using this method of description in a computer. These are:

1) It is difficult to get agreement on the precise items occurring in a "typical" case.
2) It is difficult to scope the range of all the items in the general category "plausible."
3) It is difficult to encapsulate the ability of the human to categorize a large variety of different items as plausible alternatives in a specific case.
4) It is difficult to emulate the ability to make correct identification, based on having available only parts of the items.
5) It is difficult to provide, without a large amount of processing, the ability to recognize objects regardless of size and orientation.

These problems are intriguing, because humans seem to solve them so simply and so well. Wall decorations could be art or plates or mirrors. The chairs need only be partly visible. The identity of objects is not lost when passing by them.

This kind of generalized capability is beginning to be realized in neural net machines, and that type of machine may be the only practical way to cover all the possibilities needed for the frame approach. However, an attempt to use this kind of structure to organize knowledge in the standard system can always help clarify details of the design and operation, and can be very helpful in constraining expectations.

The blackboard organization of knowledge-based systems is important because it provides a natural motivation for the use of distributed processing. This type of system is fully discussed in Nii [76]. The blackboard concept involves the use of specialized knowledge sources which communicate through a common data base. In the original Hearsay and HASP implementations, shown in Figs. 7 and 8, the knowledge sources were rule sets, each using the same data base as the others. Fig. 7 shows the basic system structure, and Fig. 8 shows the flow of information between levels of expertise. These instances also used a centralized executive, which operated the knowledge sources according to global rules of priority and the specific type of knowledge needed next. In both systems, the key operation was generation of a current best hypothesis, following the functional flow shown in Fig. 9. This provided the filter through which events were passed, and could be confirmed or denied by future events.

The Hearsay system attempted to analyze speech. The knowledge sources and their organization reflect this. They are organized in a hierarchical fashion, with a sound analyzer at the lowest level and a phrasal analyzer at the highest level. The levels shown reflect the hierarchy used in linguistic studies. Decoding sound to determine its meaning involves decoding acoustic waves to locate segments of the phonetic alphabet, using whatever values a preprocessor finds for selected characteristics. These can be frequency-based, as in the case of "formants" (characteristic frequency combinations for specific phonemes), or time-based, as in the case of typical transitions between phonemes. Above this, specific combinations of features are joined into typical phonemic segments of words. Combinations of phonemic candidates are joined to form candidate words, creating candidate syllables along the way. The candidate syllables are joined at the lexical level by using knowledge from the phrasal level, which has access to semantic information. This is shown in Fig. 8(a) by the addition of the conceptual level. This provides the hypotheses that are to be tested to finalize the decision as to what was said. It is important to note that a final selection must be delayed until the activity of all of the knowledge sources has been completed. That is, the combinations at the lower levels cannot be reduced to one (1) until a "consistent" (or "plausible") phrase has been detected. Therefore, feedback must be used to assist in the final selection at all levels, and may involve reexamination of raw data to see if a new combination can be justified that better supports a probable form for the utterance. The system must therefore suspend final judgment until the knowledge sources have reduced the possible form to only one choice by looping several times through the facts, with modified hypotheses to be tested.

The same design philosophy is reflected in the HASP/SIAP system, shown in Fig. 8(b). In this case, the levels in the hierarchy are determined by the content of the information handled. In this design, the lowest level encompasses signal detection and signal processing, a functional activity similar to

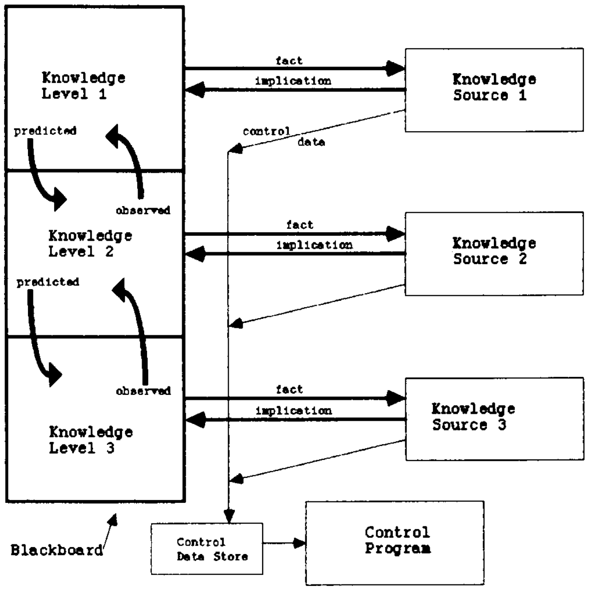


Fig. 7. Basic blackboard design.

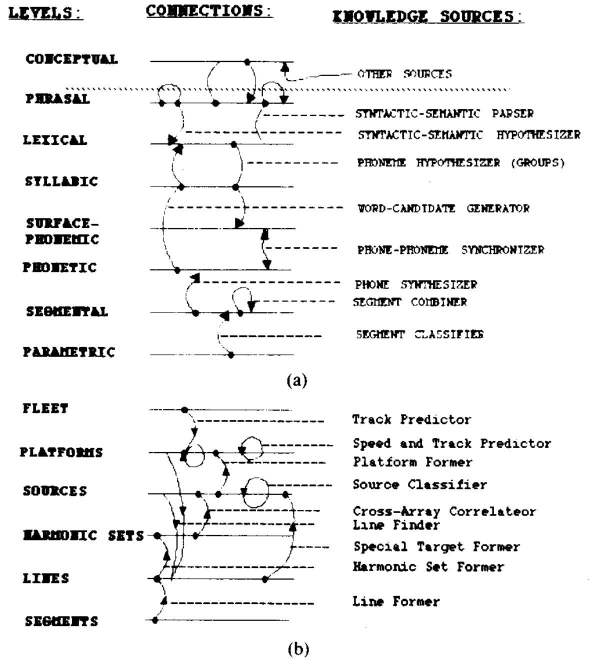


Fig. 8. Hearsay and HASP/SIAP blackboards. (a) Hearsay-II blackboard and knowledge sources. (b) HASP/SIAP blackboard and knowledge sources.

the acoustic processing activity in Hearsay. These functions are shown separately as the segments, lines, harmonic sets, and sources levels. Within the levels, the same type of selection process occurs as in the Hearsay system. The segment layer extracts groups of acoustic characteristics which might correspond to lines from emissions from typical sources. Potential lines are grouped into families at the level of harmonic sets. The harmonic sets are associated with specific sources at the level of platforms. All of this is coordinated and validated by information supplied at the fleet level. The fleet level handles the most general information, and provides hypothesis formation information from outside sources. Incorporation of other sources at the highest level allows anticipation of the appearance of a source, for example, and this allows processing to be sensitized to the expected emissions. This latter facet of the behavior emphasizes the fact that the effectiveness of the system is strongly dependent on the accuracy and flexibility of the models which derive the hypotheses transferred from a higher level to a lower level to provide pruning, or to drive a more detailed analysis of data previously connected.

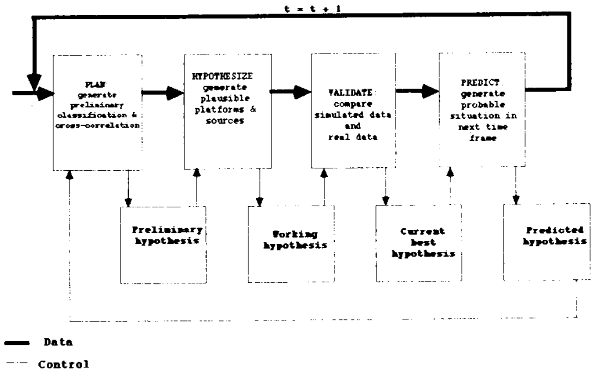

t = t + 1
PLAN generate preliminary classification & cross-correlation
HYPOTHESIZE generate plausible platforms & sources
VALIDATE compare simulated data and real data
PREDICT generate probable situation in next time frame
Preliminary hypothesis
Working hypothesis
Current best hypothesis
Predicted hypothesis
Data
Control

Fig. 9. HASP design. One global data structure contained a hypothesis about the situation. After each cycle of the plan, hypothesize, validate predict operations, the hypothesis changed states. These states were called the preliminary hypothesis, the working hypothesis, the current best hypothesis, and the predicted hypothesis.

The cyclic operation of HASP is shown in more detail in Fig. 9. In a given time interval, the operation starts with test of a prediction. This is the link between the "predicted hypothesis" block and the "plan" block. The plan block encompasses all of the analysis activity needed to verify or discard the selected hypothesis. This results in the generation of a refined or preliminary hypothesis concerning the types of sources. These are validated or discarded on the basis of comparison with data provided by simulation or processing of real data. The result of the data comparison test is then used to generate a new current best hypothesis as to the sources(s). The current best hypothesis is then used to predict what will happen in the next unit of time. If all goes well, the "predict-and-test" operation causes convergence on a single answer for the identity of any source of emitted sound which has been detected.

The major design issue in the Hearsay and HASP systems concerns activation of specific knowledge sources when an event occurs. The executive program, which controls activation of each source in turn, must have accurate knowledge of which sources are interested in what data, so that the appropriate knowledge source is scheduled when a fact is inserted. The same executive must have heuristics for dispatching knowledge sources in such a say that the more important events are processed first, so as to minimize the number of times the loop must be traversed and to minimize the number of retractions required because of the fact that conclusions generated by one knowledge source require that previous conclusions by other knowledge sources be revised.

In the examples cited by Nii, the types of knowledge used are visualized as being organized hierarchically. That is, semantics represents a higher level of knowledge than phonetics, and fleet knowledge is at a higher level than line knowledge. In any case, the organization takes into account the interaction between the types of knowledge such that knowledge at the higher level can be used for two purposes: to predict future events and to confirm current conclusions. The key element, however, is the independence of the knowledge sources, not rank. Each source is a specialist in a narrow domain, much like human specialists. This isolation into specialties makes this architecture ideally suited for implementation by means of the object-oriented paradigm.

It is not necessary to use the type of implementation described by Nii, however. In general, each of the knowledge sources could be a different machine (specialized, if desired), and communication could be by broadcast over a bus (or buses), with all relevant data captured and stored locally. That is, the system could be implemented in a way completely in accord with the rules of the object-oriented paradigm, as shown in the example discussed in Section VIII. Each level may also have buses to the other levels. Message traffic can be determined by the kind of data needed by other machines. In a dynamic system, this can change as a result of each machine registering to receive specific types of data from a distribution center, thus permitting an interesting adaptability to be achieved. One feature common to all of the systems so far implemented is adaptation of the data analysis to expectations. The system maintains a set of hypotheses, based on its world model, and uses these to focus the data-analysis effort on expected events. Actual events or facts are then used by all knowledge sources to revise plans and hypotheses. The only difficulty in this type of approach is control of the number of cycles needed for convergence. The looping operation must be executed once for each new or revised fact. The problem is in trying to determine the maximum number of loops needed for each contribution of each knowledge source to circulate through the system and be digested by all knowledge sources such that no new individual contribution triggers further changes. No method is yet known for predicting convergence or stability in such a system.

V. Problem Selection

When it is decided that AI techniques should be investigated, the first rule of problem selection is to choose a problem that appears to be too simple. It will become clear, as the implementation proceeds, that it is sufficiently difficult. It will be made especially clear when the prototype system is demonstrated to a group of experts. Many comments will be offered about all of the things it does not do. Within this general constraint of staying with "simple" problems, some guidelines are described below.

The most obvious situation in which AI techniques apply is in the capture of the thinking of a recognized expert, especially in the case where the expert is about to be lost. The opportunity to capture such expertise in permanent form in a processor so that it is usable by others is compelling. In a similar way, the ability to make the scarce resource of expertise more widely available is an excellent motivation for development of systems to assist in medical diagnosis or in making financial decisions. Extending the reach of the expertise of a good teacher is a type of situation that occurs in teaching, and this has given impetus to the use of AI techniques for computer-aided instruction.

Beyond these simple cases, however, the decision is not so clear-cut. Often, it seems to be "because we want to do AI." This is not considered the best approach. A possible set of guidelines which provide an alternative is as follows:

1) The process being investigated is performed in a heuristic manner by a human. That is, the solution depends primarily on judgment, not on calculations.
2) The process requires searching large data spaces, thus being indicative of a complex combinatorial problem.
3) There is no unique or "best" solution, but a set of acceptable solutions. In addition, judgment is required to determine which solution, within the set of solutions, should be used. This may involve generating and testing multiple hypotheses in order to compensate for missing or uncertain data.
4) It is desired to separate the data base from the executable code, so that the data base can be easily modified as a result of corrections or additions.
5) It is desired to capture, in permanent form, the reasoning and/or reaction process involved in arriving at a solution, so that reliable solutions can be generated for situations or cases which are different from previous situations or cases.
6) The process being investigated requires the modeling of human behavior.
7) The development process requires that several candidate designs be quickly outlined for comparative purposes (rapid prototyping).
8) It is desired to have easy and natural communication, with adaptability, between the user and the system.

It must be remembered that the von Neumann machine is most useful for high-speed, repetitive operations. It is therefore an ideal source of assistance in tasks which require this ability. The need to account for details which otherwise might be overlooked or the need for rapid and tireless application of routine analysis to large masses of data (a Turing machine, but with the use of heuristics rather than algorithms alone) indicates a task which is a prime candidate for the use of AI technology.

VI. Available Tools

The following sections discuss the software and hardware found to be most useful in developing knowledge-based systems. It is important to note that none of the tools described below are directly usable in real-time control problems. In those situations where the problem involves replicating expert behavior in a real-time problem, such as control of a chemical or petroleum plant, an additional issue of real-time data collection and response must be addressed. This requires the use of tools and technology being developed by GigaMOS, Inc. (the PICON system) and Gensym, Inc. (the G2 system).

These tools and systems are unique and cannot be integrated with the tools discussed below.

A. Software Tools

The tools available can be roughly divided into the two classes cited above: those primarily concerned with rules and those concerned with general knowledge units, including rules. The former is exemplified by OPS5 (Carnegie Mellon University) and Nexpert (Neuron Data, Inc.). The latter is exemplified by KEE (Knowledge Engineering Environment by Intellicorp) and ART (Automated Reasoning Tool by Inference Corp.)

Rule-based systems focus on selection of rules whose conditions are satisfied by the current set of facts, and "firing" new rules when a new fact appears, or is "asserted." The Nexpert system is somewhat more sophisticated than a simple rule system manager, in that it provides a lot of graphics and a very friendly user interface, coupled with very effective support for tracing rule dependencies. The Nexpert system also allows rules to be grouped into sets which are used only in a specific situation, thus adding a type of frame capability.

All of the available tools include some form of inference engine. In the case of such general-purpose systems as ART, KEE, and Nexpert, the environment provided also contains tools for handling frames (as unique objects), semantic nets (as dependency lists and/or a hierarchy of inherited traits), and procedures. Among the extra tools provided is a means for retracting all previous decisions affected by a change in knowledge (called a truth maintenance system). The KEE and ART systems also provide a means for experimenting with facts and situations to determine the viability of alternative interpretations of facts, or the effect of adding new rules, without disturbing the current system; this is called an alternate worlds investigation tool. In any case, the general-purpose systems represent an extension of the simple rule-based system and can therefore provide examples of operation for all paradigms and knowledge representation techniques.

In a any rule-based system, rule activation may proceed in a "forward-chaining" manner as shown in Figs. 10–13.

The example chosen is taken from the tutorials used for the KEE system. There are variations in terminology and syntax among the various systems, but the basic procedure is the same in all systems. The forward-chaining procedure is started by the change in state of some fact in the system, or by the insertion of a new fact. In the case of KEE, both cases are covered by the "assertion" operation. As a result of the assertion, the procedure described in Fig. 10 is executed. Steps 2 through 5 represent the sequence of activity represented by the term *forward-chaining.* Step 3 is taken because the fact represents new information, and the independent nature of the rules in the rule base requires examination of all rules. Each rule is examined to see if its premise (the "if" part) is satisfied as a result of the existence of the fact. If so, step 5 is taken, and any new facts resulting from execution of the operation(s) specified in the assertion clause (the "then do" part) are added to the knowledge base as new facts. This

Forward Chaining: Example

REASONING PROCESS:

When a fact is asserted:

1. If the asserted fact is already true, nothing happens.
2. If the fact is *new information* it will become part of the knowledge base.
3. Rules in the specified rule class that have a fact in their *premise* matching the fact are "tickled" (identified).
4. "Tickled" rules are tested individually. If all of the facts in the premise of a rule are true, it is added to the forward-chaining agenda (list of active rules).
5. As each rule is removed from the forward-chaining agenda and applied, its assertion (or assertions) become part of the knowledge base.
6. If the *assertion* of an applied rule causes the *premise* of another rule to be completely satisfied, forward chaining continues on the new assertion.

Fig. 10. Forward-chaining, chaining procedure.

Forward Chaining: Example

Forward Chaining can be invoked by using the keyword 'assert' with a rule class:

```
(ASSERT 'well-formed-formula  'ruleclass)
           ↑                  |→ causes reasoning to start
  basic assertion: 'add          in this class
```

For Example:

```
(ASSERT
    '(the entree of dinner is lasagna)
      'choose.wine.rules),
```

where the fact being asserted matches a factive phrase in the premise of one or more rules ("the x of y is z").

Fig. 11. Forward-chaining, activation.

Forward Chaining: Example

choose.wine.rule immediately affected:

```
 (if (the entree of ?meal is   ?entree)
                    dinner    lasagna
   (the color of  ?entree is ?color)
                  lasagna     red
   (?wine is in WINES)
    hearty-burgundy (deduced from next clause)
   (the color of  ?wine is ?color
         hearty-burgundy  red (because it is the only one
                              in WINES with red; "hidden" rule)
    then do
    (a potential.wine of ?meal  is ?wine )
                         dinner    hearty-burgundy
```

Fig. 12. Forward-chaining, initial step.

Forward Chaining: Example

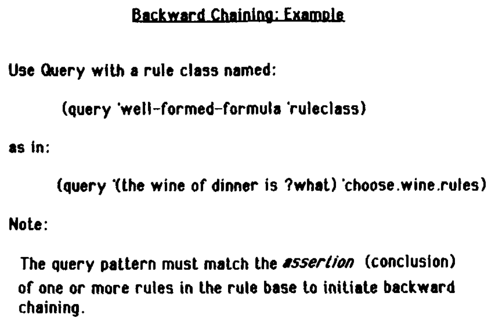
this "triggers":

```
(if (a potential.wine of ?dinner is  ?wine)
                         dinner      hearty-burgundy
   (the status of ?wine is  on.hand)←——————┐
                  hearty-burgundy          |
                                  ┌────────────────────────────────┐
                                  | "status' = attribute of object |
                                  | 'hearty-burgundy', and         |
                                  | 'on.hand' = one of its legal values |
 then do                          └────────────────────────────────┘
   (the  wine of ?dinner is ?wine )
 Note the implied 'meta-rule' that the color of the wine and the entree
        should match
```

Fig. 13. Forward-chaining, final step.

causes step 3 to be repeated, and steps 4 through 6 as well, if appropriate. This execution loop continues until no more rules are active (the agenda of rules to test is empty). In the example shown in Figs. 11–13, it is desired to find out what wine is available which is suitable for consumption with a dinner entrée of lasagna. The rules capture the heuristics such as the general rule that a strongly flavored red wine should be consumed with a strongly flavored red entrée such as lasagna, and that the desired type of wine should be among those on hand. These heuristics are captured in a knowledge unit called the choose.wine.rules. In this case, the assertion phrase matches a factive phrase in the rule shown in Fig. 12: "the entree of ?meal is ?entre," where the ? indicates a variable which may have any one of a group of values. In this case, "dinner" and "lasagna" are acceptable. The system then proceeds to test the rest of the premise, to see if it is totally satisfied. In this case, lasagna is found to have the color red, and "hearty burgundy" is found in the group of objects called WINES. The action which accomplishes this assertion has been labeled a hidden rule, because it is implicit in system operation; the system automatically examined the object class WINES and returned all objects with the attribute "red." The result of this sequence is that the fact in the assertion clause of this rule can be asserted, so that a possible wine selection is indicated. This assertion causes the rule in Fig. 13 to be tested. The sequence of tests on the phrases in this rule indicates that there is some hearty burgundy on hand, so it is selected, thus completing the search, since there is no factive phrase of the form "the wine of ?dinner is ?wine" in any of the remaining rules.

Alternatively, the rule-based system may operate in a "backward-chaining" manner, as shown in Figs. 14–17. In this type of operation, a goal is specified, and an attempt to justify it is made by checking to see if supporting facts can be found. That is, an attempt is made to derive the conclusion from existing facts by tracing backwards through the premises of rules. Necessary facts may come from interaction with the user or they may be derived by noting other rules whose "firing" would supply missing facts. The example shown in the figures is also taken from the KEE system. In this system, the goal of the search is defined by the keyword *query*. The "well-formed-formula" (wff) is any instance of a legal type of factive phrase used in the conclusion part of rules (called the assertion in KEE). The procedure invoked as a result of the query is described in Fig. 15. The fact that the wff in the query matches the wff in the conclusion of some rule causes the potential satisfaction of each of the wff's in the premise of that rule to also become goals, secondary to the goal established by the query, which is the "top-level" goal. (In OPS5 and related systems, the nomenclature is "goal" versus "subgoals".) In the backward-search procedure, in contrast to the forward-search procedure, the secondary goals can be achieved by the derivation process ("pattern matching" in Prolog) or by asking the user. In the example, the wff matches the assertion clause of the rule which asserted the wine to be consumed with the entrée for dinner. It is found that the second phrase of the premise is satisfied, and the first phrase establishes a new goal. This goal is satisfied because of the fact that the factive


Backward Chaining: Example

Use Query with a rule class named:

(query 'well-formed-formula 'ruleclass)

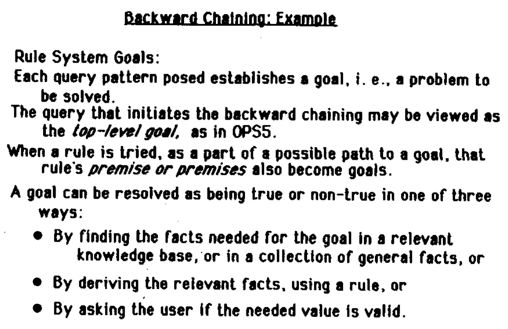
as in:

(query '(the wine of dinner is ?what) 'choose.wine.rules)

Note:

The query pattern must match the *assertion* (conclusion) of one or more rules in the rule base to initiate backward chaining.

Fig. 14. Backward-chaining, initiation.


Backward Chaining: Example

Rule System Goals:
Each query pattern posed establishes a goal, i. e., a problem to be solved.
The query that initiates the backward chaining may be viewed as the *top-level goal*, as in OPS5.
When a rule is tried, as a part of a possible path to a goal, that rule's *premise or premises* also become goals.
A goal can be resolved as being true or non-true in one of three ways:

- By finding the facts needed for the goal in a relevant knowledge base, or in a collection of general facts, or
- By deriving the relevant facts, using a rule, or
- By asking the user if the needed value is valid.

Fig. 15. Backward-chaining, procedure.

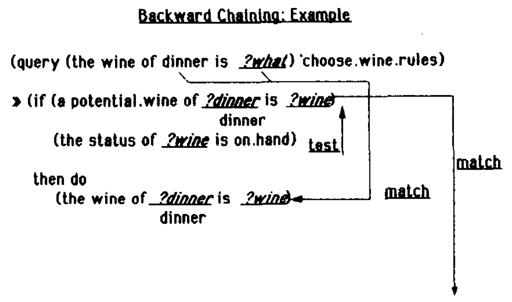

Backward Chaining: Example
(query (the wine of dinner is ?what) 'choose.wine.rules)
> (if (a potential.wine of ?dinner is ?wine)
dinner
(the status of ?wine is on.hand)
test
match
then do
(the wine of ?dinner is ?wine)
match
dinner

Fig. 16. Backward-chaining, sequence of steps.

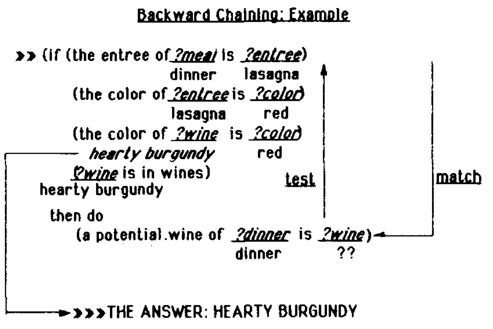

Backward Chaining: Example
>> (if (the entree of ?meal is ?entree)
dinner lasagna
(the color of ?entree is ?color)
lasagna red
(the color of ?wine is ?color)
hearty burgundy red
?wine is in wines)
hearty burgundy
test
match
then do
(a potential.wine of ?dinner is ?wine)
dinner ??
>>>THE ANSWER: HEARTY BURGUNDY

Fig. 17. Backward-chaining, the answer.

phrase which generated it is matched by the assertion of the rule shown in Fig. 17. Setting out to satisfy all of the goals created by the wff's in this premise results in the discovery that there exists a wine called hearty burgundy for which all necessary conditions are satisfied, which allows the desired answer to be generated.

Most tools provide a means for doing either of the operations illustrated above to some degree, but the sophistication and ease of use varies widely. The more sophisticated, such as Nexpert, ART, and KEE, provide automatic switching between the search mechanisms. This is accomplished in several ways. It can be done by causing the system to switch in accordance with an agenda, by establishment of new worlds which use modified rules and search procedures, or by noting that assertion of an additional fact would cause a particular rule to be enabled. It can also be done by other techniques, such as querying the user. All of these features make the rule system more flexible, and are especially useful during the design phase. They are obtained at the cost of increased bookkeeping, to provide a trace of all linkages, both forward and backward, between rules in each rule set. Therefore, they require larger memories and cannot be easily implemented on the PC type of machine. However, provision of these features also allows automatic notice of the impact of changes in individual rules or in rule grouping. This is a feature not available in the systems available on the small PC's or in the public version of OPS5. Sophistication is also added in the larger systems by provision for the extensive use of confidence factors and "fuzzy" logic. The use of confidence factors allows path activation (by selection of the next rule to fire) to proceed in accordance with a numerical estimate of the degree of certainty in assessing a cause and effect relation between facts. It also allows the final conclusion to be limited to a set of possibilities with relative likelihoods. The fuzzy logic concept recognizes that specific "things" may belong to more than one class, inheriting from all parents. It also provides a way to account for the fact that there may be multiple causes for events and the fact that there exist "uncertainty intervals" in many processes. These concepts have been treated in both mathematical and common sense ways, as described by Negoita [70]. There are currently no firm guidelines to guide a choice of which approach is appropriate in any given case.

The more powerful tools provide both powerful graphics support and a large set of tools for development. They include systems for automatic truth maintenance, so that any and all conclusions or actions voided by a new fact can be retracted. The KEE and ART systems, for example, provide a tool for experimentation, called worlds, which allows investigation of the effects of changing facts, rules, or procedures, as noted above. This construct allows extension of the rule sets by merger of worlds, where the system supplies feasibility indications. Because of their sophistication, these tools are recommended for use in cases where a reasonably large system, subject to evolution and requiring interaction between several sources of knowledge, is to be implemented.

Both the KEE and ART systems provide knowledge structures based on the frame concept, and both have been rewritten in CommonLisp to make them portable across a larger variety of systems. The KEE paradigm is especially interesting because it encapsulates all items in frames, so that they can be tested as independent objects, and then disciplines all interactions to follow the object-oriented programming paradigm. In the object-oriented paradigm, every object belongs to one or more classes. Each object inherits qualities, rules, and procedures from the classes to which it belongs, as well as having its own unique qualities, rules, and procedures. To this extent, it reflects the semantic net concept. However, each object interacts with all other objects by sending and receiving messages—there is no such thing as a program or subroutine call in the normal sense of these terms. In this respect, it follows the paradigm used in implementing Smalltalk (copyright by Xerox) and LOOPS (copyright by Xerox), the original object-oriented programming systems. The Flavors system of Symbolics is available for this type of work on Symbolics machines. Other suppliers are using CommonLOOPS, the CommonLisp version of LOOPS. Thus, CommonLOOPS is likely to become dominant, and should be considered when it is desired to program a unique system which requires portability.

Current recommendations for selection of an appropriate environment for the development of knowledge-based systems tend to favor KEE for systems in which evaluation of situations, prediction of behavior, and execution of procedures are primary considerations. The ART system tends to be favored in cases where pattern-matching and search operations are primary. These recommendations are based on the fact that KEE provides a somewhat more sophisticated capability in the use of frames and semantic networks and has a reasonable rule system with truth maintenance and an extensive graphics capability. This seems to result from its origin as a tool for geneticists. The ART system is oriented somewhat more to the programmer type of user. It provides an excellent pattern-matching capability and very good tracing of reasoning sequences. The ART system is the source of the worlds concept, which allows testing of the effects of rule changes without disturbing the current system. It reflects its origin as an extension of the rule system in OPS5, and is well suited to situations in which symbolic logic is used heavily, as in searching operations.

For situations in which the domain is smaller and the activity reasonably restricted, the Nexpert system is recommended. It has a well-designed system for grouping rules by the situation in which they are used, an excellent system for graphically browsing through a network of interdependent rules, and an extremely simple user interface. However, it requires the use of a graphically oriented system, with bit-mapped graphics, such as the Macintosh.

The cost of the tools ranges from nothing for a public version of OPS5 for the VAX to about \$50 000 for the first copy of either ART or KEE, with Nexpert somewhere in the middle (about \$7000). The relative power of the tools is roughly correlated with their cost, as indicated above.

Beyond the tools themselves, it is important to consider the issue of language used and the ability of the system environment to support languages other than that used by the tool. The language used by the tool is important because it always becomes necessary to write new procedures to augment those supplied. For almost all tools, this means that the knowledge engineer must be fluent in Lisp. Support of other languages is important, so that existing code can be executed during operation of the system, for the sake of reducing the cost of implementation. The latter situation occurs when the knowledge-based system is to be used as an adjunct of an existing system, and the existing system already has extensive code for modeling and simulation. Easy access to the existing code becomes mandatory, in this situation, and is becoming a common feature in all systems. However, the ease with which other software can be "spliced in" dynamically varies considerably. One example of a good mechanism is provided by the operation of Nexpert on the VAX. This tool makes good use of the "virtual image" feature of VMS to allow programs in Oracle, C, or any other language to interact dynamically in a manner transparent to the user.

B. Hardware Tools

The hardware systems available to run the major software systems include dedicated Lisp machines and standard machines running a dialect of Lisp or C.

The dedicated machines belong to two classes: workstation and mainframe. The workstation machine includes the Xerox and TI families. The mainframe type of machine is built by Symbolics, Inc. The cost varies from about $25 000 for the Xerox machines to about $100 000 for the largest Symbolics machine. The throughput varies roughly as the cost of the machine, but the concept behind lower cost machines is that high throughput may not be necessary. In fact, the extensive on-line memory and large mass memories now available on the smaller machines make them quite adequate for many tasks. This fact, coupled with the fact that all of the machines, regardless of price, support only a single user effectively, makes the smaller machines an attractive alternative. Use of the TI machine is made even more attractive by the possibility of using the TI "Lisp machine on a chip" as a part of a delivery vehicle. TI now supplies a Lisp processor in a 1/2ATR configuration, suitable for use on military aircraft, so that there is now credibility to the concept of a deliverable system for use in difficult environments.

Most of the mainframe and workstation machines provide the capability of running the same knowledge-engineering software, but at less speed in the workstation. The problem is that the stack processing and garbage collection functions required for the Lisp language create a heavy load for existing machine designs.

An even cheaper low-end capability is becoming available on personal machines. It is now possible to transport sophisticated software such as Nexpert to these machines, so that the newer versions of the IBM PC and the Macintosh can now be used as AI workstations. The IBM PC can, in fact, be modified to become a very powerful workstation in CommonLisp, so long as graphics are not needed, by adding a special board (the Hummingboard), supplied by Gold Hill Computer. The Macintosh currently can be used for the Nexpert software, and may soon be usable with other software, with the release of the Coral Corporation's version of CommonLisp.

Both the workstation and the personal computer alternative allow early infusion of AI technology into a commercial concern at low cost, with low risk.

VII. Implementation Issues

Given that a knowledge-based system is judged useful, and that the role of the system has been decided, there remain three general areas of concern in implementing the system. These are

- knowledge acquisition
- knowledge representation
- system architecture.

These topics are discussed separately in the following paragraphs. Because they are somewhat interrelated, the discussions overlap.

A. Knowledge Acquisition

The knowledge acquisition task can consume 50 percent to 80 percent of the total effort of implementing a system, depending on the amount and type of knowledge to be elicited. There are two sources of problems in this process. First, as noted above, the expert often cannot articulate reasons for his behavior in the form of standard rules. Second, the expert typically does not have time available for extensive interviews, especially because his expertise is in short supply. The first problem evidences itself when the expert gives a set of rules and then observes the results. These are usually unsatisfactory on the first pass, and the rationalizations originally given can even be counterproductive. For example, the expert may visualize using facts stated in the input data, but they turn out to be implicit, rather than explicit. That is, the expert may do some "preprocessing" of which he is unaware. The quality of the rules given verbally is thus about the same as the quality of those given by a native speaker of a language when he attempts to justify the forms used in his language. The fact is that they are just what is done by custom, not because they are necessary. But the native speaker feels that a more universal and logical reason should be given, when asked why.

Therefore, as a general rule, satisfactory completion of the knowledge-acquisition task will require careful observation by the knowledge engineer of the behavior of the expert in different situations. This must result in careful notations of changes in the environment to supplement answers to questions asked of the expert. The discovery procedure includes the use of videotapes plus notes on index cards, the technique used by linguists in field work in an unknown language. The discovery process must then be followed by constant observation of the emerging expert system by the expert.

In general, the knowledge acquisition process therefore has anthropological aspects and linguistic aspects. Above all, it involves the general problem of trying to duplicate human behavior without knowledge of what significant features the human responds to and without conducting a sensitivity analysis to determine them.

B. Internal Knowledge Representation

The problem in analyzing the knowledge of the expert can be restated as resulting from the fact that the knowledge engineer has to codify the behavior of a black box whose inner workings are not visible. This problem is made worse by the fact that the activity to be codified is not verbal in nature, and cannot be easily translated into verbal form—sometimes it cannot be translated at all. Successful analysis involves knowledge of significant features whose nature and identity are unknown at the start, and which are amenable to rigorous description only by execution of a sensitivity study too complex to undertake in the time usually available. Beyond this, there are issues of the form and process involved in cognition and concept formation by which the mental models are composed. The nature of these facets of knowledge represents an issue which has not yet been resolved. The fact of the existence of items such as an idealized conceptual model (or prototype) seems obvious, but their structure and rationale have yet to be adequately defined. Even if they could be defined, it is possible that they may not be accurately realized in existing machines, because of the profound differences between the machinery of the brain, as compared to existing computers.

Unless this is recognized, implementation of a satisfactory system will be difficult. It is clear that the knowledge acquisition process transcends simple logical operations and requires much more effort than simple interviews. The problem is complicated by the oft-imposed constraint that the model developed for the knowledge of the expert must be composed only of the elements of traditional logic.

C. External Knowledge Representation

There is a significant problem in choosing the framework in which the knowledge of an expert is to be inserted. This problem arises from the fact that the result must be usable in a sequential machine which has only two dimensions of interconnection, which has very few connections between components, and which is restricted to sequential operation in any one processor, so that human activity cannot be easily described, if at all, in the simple system of linguistic and logic structures available. In addition, humans operate with partial data and with degrees of uncertainty not easily captured in either the language of the machine or its logic. It is generally true, therefore, that a mixture of methodologies will be needed to provide a reasonable emulation of human behavior. This means that the system will make use of all three of the basic structures described above. In all cases, however, it must be remembered that success depends on the extent to which the domain of the problem can be limited and the behavior of the system constrained to a small, well-defined set of possibilities.

The frame structure is useful in formulating typical situations and exposing the assumptions associated with them. This is important in systems which must advise on appropriate behavior. For this purpose, the fact that the frames can include predicted behavior and methods for generating hypotheses makes them a powerful tool. This is over and above the ability to minimize the number of rules which must be checked when any designated type of event occurs.

The frame structure is especially attractive because it focuses on the context in which behavior occurs. This means that a new frame can be chosen when the context changes, allowing a rather natural change in the set of rules used. It is also useful because it is natural to include the actors creating the situation, with their expected behavior. Standard behavior can be included in the form of a script, and this can be used to guide the selection of rules and hypotheses. Scripts, in turn, can be derived from standard missions or goals, thus creating a strong and well-motivated causal link between all of the elements of the system, as well as allowing deduction of facts and actions not explicitly stated (see Schank [78]). When all of the elements of the frame structure are considered together, the frame is also very helpful in indicating the actions required to retract all the effects of a conclusion when events require a change in beliefs.

The problem with using frames is that, often, no real situation is exactly the same as the "typical" situation, so a lot of thought must be given to the variation allowed within a given situation before a change in situation must be postulated. The deviations from "typical" or "normal" which must be accommodated in frame-based systems suggest that the use of fuzzy logic may be required, but the kinds of variation allowed, and their extent, can be difficult to enumerate.

The tree structure, on the other hand, can be used to show the specific nature of relationships between elements of the knowledge. It can also be used to search for the best action to take next, or to help decipher a sequence of words, as in augmented transition networks, where each word restricts the set of following words which are plausible. However, the evidence that humans do not search sequentially through a set of rules must be kept in mind. Evidence now available points to a single-step process for recognition of a stimulus and selection of a response. Humans seem to operate sequentially only in speaking and moving. All other operations appear to be done in parallel, in very few steps. The point is that encoding knowledge in a tree form is reasonable for showing semantic relationships, but it is a difficult form for defining search or classification operations, because it introduces artificial, linear constraints. As a result, encompassing a large set of cases is difficult, and changes are difficult. This means that refinement of knowledge is difficult, so that the sequence of changes inherent in the process of progressing from novice to expert are difficult to enable. Therefore, tree representations of knowledge, including state diagrams, are a very weak model to use in attempting to emulate human behavior. Attempts to use this type of structure in a von Neuman machine graphically illustrate the limitation of the computer, as compared to the brain.

Rule sets are satisfying for modeling simple decision processes. To allow rapid development and test, plus easy maintenance, the number of variables must be small, and the combinations few. Rule-based systems are easy to build—rules can be generated at the rate of several hundred per month. It soon becomes clear, though, that interactions between the rules are too complex to easily trace and control without special tools, and may be difficult even then. It will always be necessary to introduce control methods, such as frames, and tracing methods, such as semantic networks, to avoid disastrous side effects when changing the rules. Therefore, use of rules alone should be undertaken only for very small problems.

There is evidence that the ease of system implementation varies in proportion to the complexity of the tool. That is, the designer can quickly build a system which uses only a set of unordered rules. Building a system which uses sets of related rules with controls for selection of sets or rules by means of semantic relationships takes longer. Expanding the knowledge structure to include frames causes the implementation to take longer still. It has also been found, however, that a large staff is required to maintain a large, unordered rule set, because of the extensive debugging required whenever changes are made. It is likely that most problems are sufficiently complex to require the extra effort involved in using the complex tools because most problems are not as simple as originally assumed. This is the reason for the rule that the selected problem should appear to be overly simple. Things are rarely as simple to do as they seem to be.

D. Language

There are cases in which the system is to be entirely constructed by original programming, not by using existing tools. In this case, the user can select the language felt to be most appropriate for the kind of problem and/or the approach selected. When this occurs, there are several choices for doing object-oriented programming, and one major alternative for rule-oriented programming.

The major choices for doing object-oriented programming are Smalltalk-80 and CommonLOOPS. Smalltalk-80 has the advantage that it is available on the Macintosh and the Tektronix workstation. CommonLOOPS has the advantage that it is the portable paradigm for workstations and mainframes. All of the various paradigms, including Flavors, are discussed in Stefik, [80].

The major alternative for rule-based systems is Prolog. This language is favored for declarative types of programs because it has a rigorous formal logic, good pattern matching, and good resolution rules. Examples of problems in which Prolog has been useful include parsers for computer languages and relational data base languages. The basic criterion indicative of its value is that the knowledge is wholly factive in nature.

In general, however, the tools created in Lisp have broader applicability, because different kinds of knowledge can be modeled and coordinated. Furthermore, the graphics and user interface tools are more extensive and more mature.

VIII. A Practical Example

The Experimental Autonomous Underwater Vehicle (EAVE), in development at the University of New Hampshire Marine Systems Engineering Laboratory, will be used as an example of the practical use of the ideas in AI. There are several reasons for this selection:

- it involves real-time operations;
- the design blends all major tools and methodologies;
- the multiple levels used reflect human mental structures;
- the alternatives provided realistically mimic human activity;
- the implementation chosen allows evolution in a simple and straightforward way.

All of these features of the design create a unique concinnity. The most important of the tools used are the multiple blackboard and the object-oriented paradigm. This extensive use of the latest developments represents another important reason for discussing it, instead of previous projects.

In addition to the good selection of tools, however, the system design is organized to reflect a reasonable model of human activity. The reason for this is that the design is hierarchical, with separate levels handling different parts of the time domain, so that "reflex" actions are independent of "planning" actions. Further, each knowledge source and execution module operates as a coequal of all other modules, in a collegiate type of organization. This minimizes the capability required of the coordinator and facilitates implementation in other architectures, such as parallel processors.

The design is in the process of changing, to incorporate insights gained from experiments conducted with a prototype model. Therefore, this description represents a snapshot of the design in the form it is currently taking.

The multiple blackboard part of the design is based on the dual-blackboard approach defined by Barbara Hayes-Roth [61].

A. Missions

The EAVE vehicle is both unmanned and untethered, so that it is completely autonomous. It will have the capability to explore or map an underwater area independently, given only a statement of the goals of the mission in general terms. An example of the kind of terrain it might be asked to operate in is shown in Fig. 18. The assigned mission might be one of the following:

- exploration: map bottom and location of objects;
- salvage: locate and pick up object;
- military action: attack, decoy, or neutralize.

An unmanned vehicle allows operation in hostile environments. The use of an untethered vehicle provides greater freedom of action, and avoids potential tangling problems when obstacle avoidance is required. Total autonomy is preferred because of the limitations on the rate at which information can be transferred in the absence of a tether. Other complications avoided by the untethered design include weight, maintenance, and the problem of control of payout and payin when maneuvering.

In any case, there is an element of uncertainty in execution of the mission which prevents the planning of a precise program of actions. The instructions for the mission can only be considered to constitute a "wish list" of desired activity, with the understanding that variations will be made as the mission proceeds. These variations must include the possibility that all of the desired tasks cannot be performed, and the mission instructions must reflect this, as described below. Therefore, there must exist a reasonable level of "intelligence" on board the vehicle.

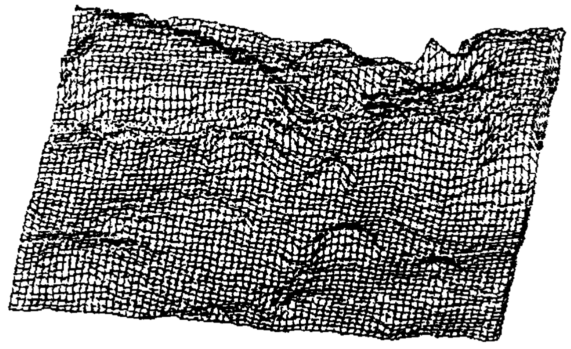


Fig. 18. Typical mission area.

The blackboard design approach is desired because of the domain neutrality it allows. Domain independence is achieved by separating the domain-specific knowledge from the mechanics of applying the knowledge. Domain-specific knowledge is encapsulated in the knowledge source, and domain-specific data are posted in the blackboard, for observation by all concerned, in any form the designer wishes to use. Therefore, while designing the system, the designer is free to implement as many knowledge sources and actions as are required to solve his problem or to allow changes to be incorporated. The posting technique used may consist of a broadcast (as on a bus) or activating a knowledge source as an object, with a pointer to a buffer. In addition, the blackboard paradigm encourages the exploitation of any modularity inherent in the design. This applies to both the procedural knowledge embedded in the knowledge sources and the data posted on the blackboard. Further, the modularity used is also flexible, so that the size of the modules can be adapted to any desired level of granularity, representing any hierarchical pattern desired.

B. Design Requirements

For an unmanned, untethered vehicle to be practical, the question of the model for its intelligent behavior must be addressed. In the standard expert system approach, the model is a recognized human expert. In the case of exploration by unmanned vehicles, there is no expert in the use of such vehicles, so the art must be learned as experience is gained. Furthermore, it is not clear what kind of expertise in exploration should be captured, given that the vehicle should behave like a human carrying out the same kind of mission. Therefore, no attempt is made to embed "expertise," in the sense normally used. Instead, the focus is on embedding "reasonable" patterns of behavior, in the sense that they reflect the activity of an intelligent human. In other words, an expert in exploration is not needed for many missions; the need is for an intelligent anticipation of contingencies, combined with a reasonable repertoire of responses. This approach is reflected in the design. The multiple blackboard and the encapsulation of individual knowledge sources as separate objects are used specifically to allow the flexibility needed to add the results of learning, as well as the need for different kinds of vehicles in different environments.

The need for flexibility is reflected in the contours shown in Fig. 18. To plan a mission completely, with detailed instructions covering all contingencies, prior to vehicle launch is impractical, as in the case of tethered vehicles. Therefore, as the mission proceeds, the path followed must bend and fold as required to reflect the actual environment. The possibilities are defined to be either "preplanned, with no variations allowed" or "tentative plan, variations accepted." The latter approach is chosen, and the planning is done in such a way as to minimize commitment on any part of the plan as long as possible, thus incorporating the "least-commitment strategy" for the path selection operations. The basic constraint adopted is that variations should maintain mission integrity, except for well-defined contingencies.

The practicality of this approach also depends on the degree to which obstacles and events can be grouped into simple classes of "typical cases." Currently, the grouping envisioned for obstacles is "wall," "edge" (or "tube"), and "block" (can include "sphere"). The set of events which may require mission abort (or may prevent achievement of all desired goals) includes such things as "fuel exhaustion" and "leak in enclosure." To allow such occurrences to be handled in a reasonable way, a suitable repertoire is embedded at each level of activity, as described below. None of this represents a procedural difference from a standard computer program, in execution. It does differ, however, in the manner in which the steps of the execution are activated and revised. It is possible at any time prior to actual execution of a step to revise or eliminate it. This allows any future commitment to be changed as needed, including further examination in a given area, reordering activity, or eliminating some tasks. In effect, a part of the program is being written as the mission progresses. In engineering terms, the designs may be compared as "underconstrained" versus "overconstrained," where the standard algorithmic approach represents the overconstrained mission execution.

The following types of activity are anticipated:

- prior to launch: task assignment and preliminary mission plan;
- during mission:
    - quick reaction to obstacles,
    - planning of existing subtasks,
    - specification and planning for new subtasks (i.e., examination of an object of interest),
    - mission revision.

To encompass these types of activity, a hierarchy of three levels of processing is established, as shown in Fig. 19. These three levels are determined by the time domain in which the activity must occur.

The types of activity required are exemplified by the cycle of search activities shown in Fig. 20. This figure presents a set of tasks representative of the type of reconnaissance mission which might be assigned to the vehicle. In this figure, the mission consists of two search operations, indicated by the rectilinear patterns, and a group of points to visit, enclosed in the large circle. The supervisor module inserted the way points for the search patterns, and a human specified the way points in the visitation group. The dashed lines represent paths

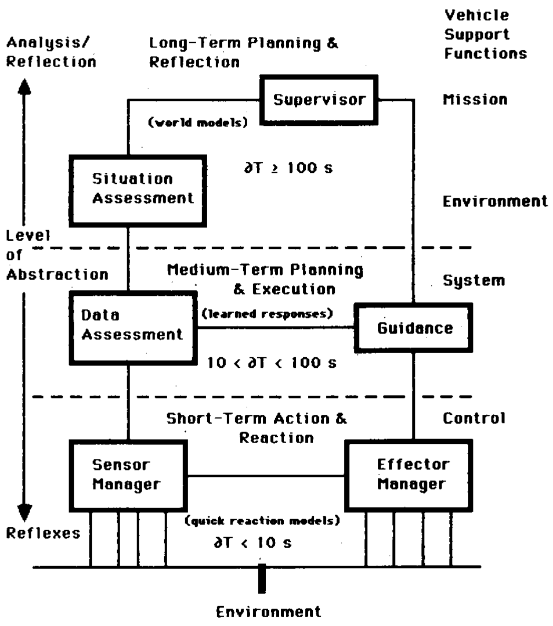


Fig. 19. Levels of activity.

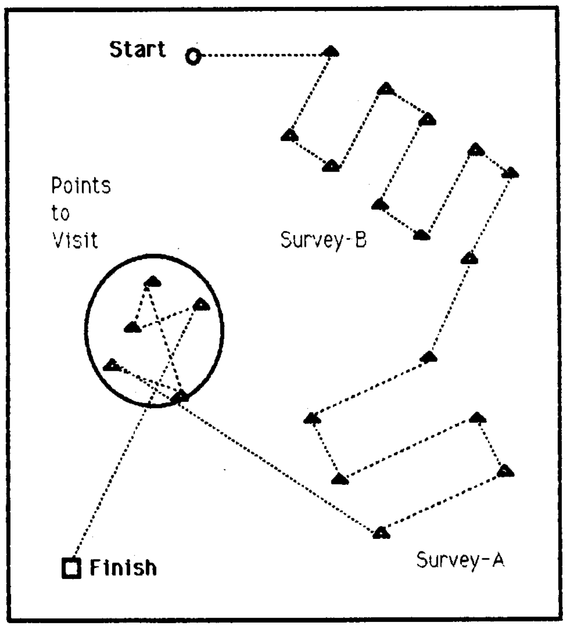


Fig. 20. Typical mission.

specified by the supervisor module. The description which follows is abstracted from Chappell [41].

At the lowest level, the quick reaction time required makes the operations comparable to the reflex actions of humans. Operations at this level thus have the nature of stimulus/response activity. The time period for plan elements, when loaded, takes into account the response rate of the vehicle, the "worst" type of obstacle, and sensor capability. This level reports status and unplanned events to the upper levels. In carrying out their normal activity, the system elements at this level make use of stereotypes such as the wall/edge distinction for obstructions. Each stereotype evokes a standard response, based on classification by the sensor. In the event that something occurs which is on an emergency list or an alert list, the vehicle can pause and await further instructions.

The middle level provides planning for the next sequence of steps, with the ability to revise paths as needed, based on the tentative plan and events occurring during the mission. Partial plans for different tasks are finalized and down-loaded to the reflex level as needed. This includes reaction to obstacles, where the reaction is to await instructions (in the event that a system problem occurs). This level thus fits together the pieces of the plan as the mission unfolds, under the constraint that sufficient energy remains. If a problem occurs, the top level is notified, and a more drastic revision is undertaken at that level.

The top level generates the initial plan and monitors activity during the mission. The function performed during the execution phase consists of a continuous comparison of the current situation with the planned situation at that point. Discrepancies are evaluated to determine if major revisions are required. If so, a new master plan is created and down-loaded to the middle level for implementation. Revision actions can also be triggered by events logged at the lower levels, such as fuel consumption. This activity can be described as a constant redetermination of the feasibility of the remaining part of the mission, given the current data.

An example of how the planning cycle works is shown in Fig. 20. The mission depicted in that figure contains three tasks:

- visit five points in a specified area;
- conduct a survey on the area covered by survey A;
- conduct a survey on the area covered by survey B.

It is clear that planning an "optimum" path is impractical, especially in view of the uncertainties implied by the nature of the mission ("survey"), plus possible vehicle problems. Therefore, any "reasonable" solution is accepted, where the term *reasonable* includes such things as fuel reserve and a nominal allowance for possible problems. However, to further ensure the execution of a useful mission, even if the initial plan cannot be completed, each of the requested tasks is given a priority, and the higher priority task is completed first, where practical. This operates in the other direction, also. If sufficient energy reserve remains and a desired visitation point is near, it will be incorporated, even if it is not in the plan. The allowance for problems can be made reasonably conservative by assuming a "worst-case" event, such as a "wall-like" obstruction which might require a large path change. Instructions to examine certain classes of objects can be executed dynamically by changing the local path. The philosophy and operation of this type of flexible control are covered in detail in the Hayes-Roth paper.

The major elements of the design can be summarized as follows:

- Exploit simple, general models of objects and slopes and responses.
- Exploit limits of maneuver capability as useful constraints for planning.

- Employ beam-directed search, where the beam is composed of typical situations and appropriate responses.
- Use least-commitment strategy for execution, postponing development of detailed plans until needed.
- Plan in detail only for "reasonable reaction period" determined by maneuver envelope and typical obstructions.
- Discard irreversible steps.
- Treat start of new operation as new mission (compartmentalize, so that each task is a separate object).
- Distribute the expertise between levels, so that each level is self-sufficient in its own domain.

The design incorporates ideas described in detail in Minsky [74] and in Agha [26]. Specifically, the concept of "cooperating agents" defined by Minsky and the concept of "actors" defined by Agha have strongly influenced the implementation of the blackboard architecture selected as the basic structure.

C. Design

The basic operational elements shown in Fig. 19 are as follows:

- temporal hierarchy: similar response—time activities in same level;
- physical separation of levels operating at different rates;
- a "loop" structure for each level, providing semiautonomous operation.

The top level is concerned with strategy and mission control. It generates the initial plan, using some mixture of an optimization heuristic and priorities of required tasks. All plans are considered to be nominal, and alternatives are specified, especially where segments may have to be deleted or modified. However, most of the operations are done prior to the start of the mission. During the mission, activity is restricted to monitoring reports, and mission planning only becomes active when the designated repertoire of contingency reactions is not adequate. Infrequent activity allows operation at a slower rate than the lower level.

The middle level is concerned with implementation in detail of the current plan, with reasonable alternatives in the event that any predefined type of object or obstacle appears, in which case it adopts a predefined modification of the plan, with a notice to the top level. It is concerned with the impact of the current activity on the nominal plans for the next activity with detailed planning for the next activity, and execution of a repertoire of contingency plans, such as modifying start and end points or deleting segments.

The middle level thus operates at a slower rate than the bottom level, but faster than the top level. The bottom level is concerned with the real-time steering of the vehicle, and with rapid response to objects or obstacles within the constraints of the available avoidance procedures. It therefore operates at the rate required to process and react immediately to data collected by the sensors provided on the vehicle.

The analysis/planning loop of the situation assessment and supervisor modules is typical of the design at all levels, and has been more fully implemented in the experimental model. Therefore, the supervisor level will be discussed in detail. The discussion will describe the difference between the current design, which is a standard Hayes–Roth implementation, and the future design. However, it is important to note that the new implementation is designed to provide independence from specific hardware architectures.

1) Supervisor Subsystem Architecture: The basic architecture of the supervisor component is shown in Fig. 21. It consists of four modules: two blackboards and two sets of knowledge sources. Each blackboard acts as a means of communication between those knowledge sources which handle the types of data normally appearing on it as indicated. However, interaction between events in the domain area and data in the control area is allowed, as indicated by the diagonal lines crossing over between the knowledge sources and the opposite blackboard.

The underlying structure of the blackboard architecture is very simple. It consists of three components: a structured data space called a blackboard, a set of reaction rules or procedures called knowledge sources, and a scheduling mechanism for controlling the knowledge sources. The structured data space may be real or it may be virtual, depending on the implementation. In any case, the resemblance to a normal blackboard results from the fact that it serves as a central repository for information potentially useful to several knowledge sources; i.e., the information is made generally available, instead of being addressed to specific modules. The dynamics of the basic blackboard system are also rather simple. The scheduler executes a three-step control loop:

1) determining which knowledge sources should be triggered in response to the posting of data on the blackboard;
2) selecting knowledge sources for execution;
3) actual execution by the knowledge source of the inference operation.

Each knowledge source potentially posts new data as a result of its execution, thus triggering a new cycle. This is the sequence used in the blackboard systems described by Nii. However, the standard approach requires that the scheduler be a "superexpert" who knows everything in every rule used by each knowledge source, plus the value of that knowledge. This complication can be avoided in two ways: by having each knowledge source examine a new fact in parallel with all other knowledge sources, using it when it is applicable, or by allowing each knowledge source to operate once per new fact. In either case, the result of the operation of the knowledge sources will be the posting of one or more new facts or desired actions. The coordination module then arbitrates between solutions with a set of rules supplied for this purpose. The new approach makes use of the fact that the purpose of the blackboard is communication, and communication can be done in several ways. The first is the broadcast of data over a communications medium, such as a bus. The second is the scheduling of each knowledge source, with a notice of the new data as its input. The first approach is preferred, because it allows all knowledge sources to be physically located in separate processors. In fact, however, a mixture of the two will probably be necessary in any reasonably compact system,

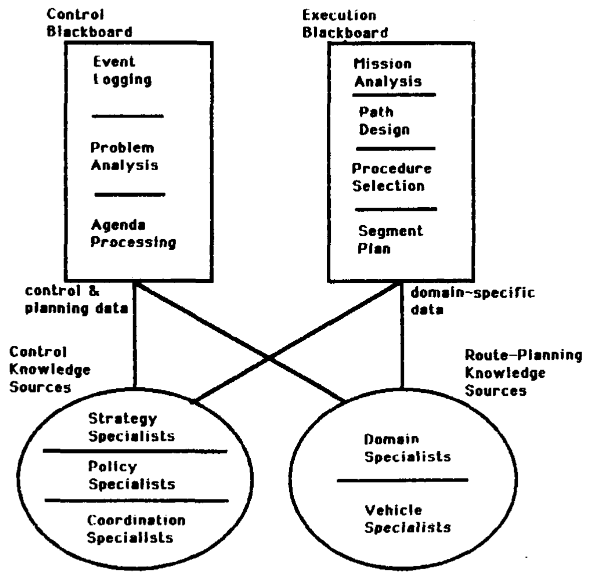


Fig. 21. Blackboard control architecture.

so that some knowledge sources will have to share a CPU resource. In either case, however, the knowledge required in the coordination module is reduced, and the operation resembles the collegiate system used by Berliner [32] in his chess-playing machine. That is, all knowledge sources are equal in their priority of firing, but the productions requested by each are not equal, if they conflict.

The fact that these variations are possible testifies to the flexibility of the blackboard architecture, and justifies its selection as a significant advance in the field of AI. Additional advantages accruing from the use of the blackboard are as follows:

- it is domain-independent;
- it can be implementation-independent;
- it provides a mechanism for superimposing a learning ability;
- it uses modularity as a basic feature;
- it is easily adapted;
- it encourages a view of the problem different from the algorithmic view.

Independence from the domain is achieved by the fact that the nature of the knowledge sources does not affect the basis design. It only affects the items posted for general consumption, and the types of activity triggered by the activity of the knowledge sources.

Implementation independence results from the fact that the communication role of the blackboard can be realized in many ways, and the machines in which the knowledge sources are embedded can be completely different in nature. In fact, it is possible to include in the network machines as diverse as neural net machines, to enhance the learning and recognition capability without requiring any changes in the design.

A good example of the way that a common bus may be used to achieve the communication function of the blackboard is given by the Berliner chess machine. This machine consists of 64 microprocessors, each of which acts as a knowledge source for a particular square, plus a Sun workstation, which contains the strategy expert and acts as the interface with the player. The 64 micros act as a collegium, in which each knowledge source has equal value. The description of each move is broadcast on the bus to all other knowledge sources. Those which are affected by the move examine its implications, in terms of threat value or advantage, and each recommends an appropriate response. The strategist compares the threat evaluations and suggested responses and selects the move to be made. The strategist, of course, has several options, including employing a new tactic. It may, for example, adopt a risky tactic, in an attempt to gain a significant advantage if the risky move succeeds. This illustrates that a focus on the **function** performed by the "blackboard" type of architecture, rather than the details of a specific method, allows a very wide variety of implementations to be visualized. A mixture of the Berliner approach and the Hayes-Roth approach forms the basis for the EAVE design.

Modularity is a basic feature, owing to the concentration on isolation of the knowledge sources in separate modules because of the independence of the knowledge encapsulated in each one. The current design fully exploits this fact in a way that allows the implementation to exactly follow the "object-oriented" paradigm. Each knowledge source is treated as a separate object, whose internal operations are unknown. For each object, the external knowledge about it consists of where it is (its address), what messages it accepts, and what messages it sends. All operations are conducted by passing messages to designated objects, which can be located anywhere. This completes the isolation required to allow a parallel computer architecture to be used.

The object-oriented design which allows the implementation to be flexible also makes the design extremely adaptable. New knowledge sources can be easily added, and the implementation used for a given knowledge source can be changed without requiring redesign of other knowledge sources or of the coordination module. Thus, it perfectly complements the blackboard architecture.

The standard algorithmic view would use the knowledge to define branches at specific points in a procedure. After commitments have been made, recovery from the prescribed course, to account for new information combined with reprogramming of the system, would be impossible unless all possibilities at all points had been predefined. The modular approach and the cooperative activity avoid this problem. The approach desired in the fact of uncertainty is the linear programming approach discussed in Charniak [42]. Many of the design issues involved, including synchronization of the knowledge sources, reflect the problems for concurrent programming discussed in Holt, Lazowska, Graham, and Scott [64].

The dual blackboard philosophy (cooperating knowledge sources) is used at each of the EAVE operating levels shown in Fig. 19. One is the blackboard on which information relative to control of the mission—the control blackboard. The other has information pertinent to execution posted on it—the execution blackboard. The posting of new data on a blackboard, either a new fact or a modification of an existing solution, is called an event. As the figure shows, knowledge sources may post to either blackboard, and they monitor both blackboards.

The descriptions which follow are based on the design described by Chappell [41], with some modification to reflect the improved design being implemented at this time.

The control blackboard contains data associated with the selection of strategy, the allocation of resources, and the coordination of knowledge sources. It is primarily used by the control knowledge sources. This blackboard also records the occurrence of internal and external events which are used to monitor the progress of the mission. The data posted here are divided into the three types shown. These are

- event logging,
- problem analysis,
- agenda processing.

Two types of events are logged: exogenous and endogenous. Exogenous events are events occurring outside the vehicle. Endogenous events are events occurring inside the vehicle. At the supervisor level, the exogenous events logged are the input of mission request prior to the mission. Problem analysis data include information about the number and types of activity required, distances, priorities, etc. Agenda processing refers to the development of a list of pending execution steps to allow all knowledge sources to examine the latest information and post the implications, plus the information required to permit evaluation and arbitration.

The execution blackboard contains data concerning the execution of the mission under constraints imposed by the control knowledge sources. This includes planning of pieces of the path (''segments'' grouped as tasks), as well as the order in which various parts of the mission are carried out. It is also used to communicate information about vehicle activities. Four types of data are identified, as shown. These are

- mission analysis,
- path design,
- procedure selection,
- segment plan.

Mission analysis data consist of information pertaining to segment identification, segment size, type of activity per segment, priority, etc. Path design information concerns the start and endpoint data, maneuvers, etc. Procedure selection information concerns the algorithm to be used to command the execution of the desired patterns, and also to estimate the time and energy required. Segment plan information is composed of the command sequence and timing data compiled from the planning activity.

The control knowledge sources are specialists in selecting the strategy to be used to complete the mission to the extent allowed by the current situation. They also specialize in communication with, and coordination of, other knowledge sources.

The route-planning knowledge sources are concerned with executing the current plan by implementing a path consistent with it, in terms of the number of segments and types required, their order, and the path between them. Their specialties therefore involve the domain and the vehicle.

Within the control group of knowledge sources, three types are identified:

- strategy specialists,
- policy specialists,
- coordination specialists.

The strategy specialist knowledge sources are concerned with planning the vehicle path so as to satisfy some heuristics for optimum use of the resources. This might include minimum energy consumption, maximizing dwell time in specific areas, or modifying activity around an object of interest.

The policy specialists are concerned with the ranking of alternatives and the selection of the subset to be used by the execution specialists on this mission. These specialists are concerned with ordering the list of pending activities and arbitrating between conflicting solutions and/or conflicting requests for access to resources. Included in this group are specialists on procedures to use in executing search patterns and in taking evasive action, as well as in reacting to emergencies and/or recovering from disturbances.

The coordination specialists are concerned with notification of each knowledge source of the occurrence of a new event, with posting of proposed solutions or activities resulting from knowledge source activity, and with maintenance of a history of knowledge source activation. These modules note the occurrence of the event, distribute the information to concerned parties, and record all responses.

The route-planning knowledge sources are divided into two types:

- domain specialists,
- vehicle specialists.

The domain specialists are concerned with the detailed planning of the path. They include knowledge sources which set the starting and ending points for each segment, locate the tasks and procedures needed, expand specific types of segments into patterns, close up any gaps in the path, and otherwise refine the design. Two types of segment requiring specific patterns are identified: the survey type and the point-group type. The first represents an activity over an area; the second represents linkage of points. These knowledge sources also generate commands required to vary the activity of the sensors in the event that different activities are required at different times, such as initiation of a special search pattern for the survey activity.

The vehicle specialists group consists of knowledge sources which represent expertise in vehicle capability and performance and which supply information needed to specify each maneuver in detail and to assess the budgetary implications of the plans in terms of total energy capacity.

Thus, the domain specialists are concerned with the details of the geometry of the mission and the procedures used to generate specific instructions, whereas the vehicle specialists are concerned with the state of the vehicle and the capability of the vehicle to maneuver as desired.

The basis input to the supervisor module is a specification for a mission, provided by the user. The output of the module is a set of directions for the areas to be examined, the actions in each area, the order in which they are to be done, and the alternatives in the event that certain combinations of events occur. This output is supplied to the next level. The supervisor module is relatively inactive during activity at the lower levels.

The basic sequence of operations is as follows: All events are noticed by the accept-input knowledge source, which distributes a notice of the event to all other knowledge sources. The other knowledge sources acknowledge receipt, and those who are affected by the event post a notice of further information to follow. The affected knowledge sources now become active. A list is made of the knowledge sources now active, and planning activity is suspended until all inputs have been received. The set of responses is examined, and a single response is selected, in accordance with the heuristics for priority, to be distributed as the next event of interest. This sequence continues until no new facts or productions are posted. In this procedure, all knowledge sources are treated as equals, but arbitration is done between conflicting outputs.

The same type of organization is used at all levels, with variations in the details covered in the knowledge sources. For example, the tentative plan is accepted by the second level as a set of requests, with estimated time and energy utilization. The plan is then examined as a group of segments, where the separable parts, such as survey A, survey B, and point-group, are examined as independent tasks. The result of this examination is generation of a detailed sequence of commands for the next segment, plus a new set of estimates for time and energy consumption, as required. The actual time and energy usage is monitored continually, so that plans for both the current and the next segment can be modified as needed. The knowledge sources available also provide the knowledge needed to initiate special sensor activity, such as special transmission or search patterns to help collect information about the nature of an object or obstruction.

The lowest level contains the highest concentration of procedures, since it contains all of the algorithms used for control of the vehicle and the sensors. Its knowledge sources and blackboards are strongly focused on detection and processing of information affecting the routine execution of commands posted by the middle level. It contains the characteristics identifying detected objects as belonging to one of the specified classes, and the appropriate reaction. In the case of obstructions, for example, the classes distinguished are edge, wall, slope, or object. The reactions distinguished are small maneuver, large maneuver, or pause. Corresponding sensor activities are provided: narrow search or wide search (vertical, horizontal, or both).

This design is unique in its combination of techniques. It uses the blackboard approach, extended to encompass a more complete form of the object-oriented paradigm, and a partitioning of system functions by response time, to permit efficient operation. As a result, the system has a unique structure, being multidimensional. Since the design is independent of hardware configuration, languages, number and types of knowledge sources, and knowledge representation within a knowledge source, it is easily extended and easily adapted for many other purposes. It differs significantly from the standard blackboard control system described by Hayes-Roth and incorporated in the experimental vehicle in the following ways:

- The knowledge sources are autonomous and operate in a collegial fashion, with the arbitrator being the first among equals.
- Events are effectively, if not actually, broadcast either to all knowledge sources or to the knowledge sources registered to receive the type of event posted.
- Arbitration takes place after each knowledge source has finished operating and not before any knowledge source has operated.
- There is a provision for the standard real-time requirements for interrupts, and for priorities to be attached to all events, including requests and alerts.
- The system is event-driven, provision being made to handle both endogenous and exogenous events.

Object functions and their operations strongly resemble the actions of the "society of cooperating agents" postulated in Minsky [74].

Current implementations of the blackboard system reflect the programming problems associated with use of the von Neumann architecture. To minimize conflict over use of a single CPU, there is a "superman" type of expert. This superexpert combines rules of importance with his detailed knowledge of all of the rules in all of the knowledge sources to select one knowledge source at a time for operation. The invocation of knowledge sources is treated another case of the standard procedure all in typed languages, with the facts used by the knowledge sources treated as variables. This means that the scheduler must know where and how all facts are used before the operation starts. This introduces a lot of redundancy and makes the system both rigid and fragile. The rigidity is because change becomes more difficult; the fragility because all possible combinations must be anticipated in the rules of the dispatching module. Changing to a structure which permits autonomy removes this problem while also eliminating the architecture sensitivity of the design.

The concept of transmitting event data to more than one knowledge source makes the design easily adaptable to other architectures, also. It can be installed in a system in which all knowledge sources are located in separate computers and communicate via a bus, or where all knowledge sources are in the same computer and each is registered to be activated when specific kinds of events occur, or any combination of the two. In the case where all knowledge sources reside in the same processor, the operation will take the form that the blackboard is physically realized in a buffer, and each knowledge source is dispatched with a message about the buffer. This way, the buffer may be distributed, with different memories for different types of events. With this change, the system has been made completely architecture-independent. Arbitration, in this case, is limited to access to the transmission medium or local resource, and to the adoption of the implications posted by a given knowledge source. This is done by comparison to the rules for strategy and tactics, and uses the implications derived by all knowledge sources. All knowledge has been

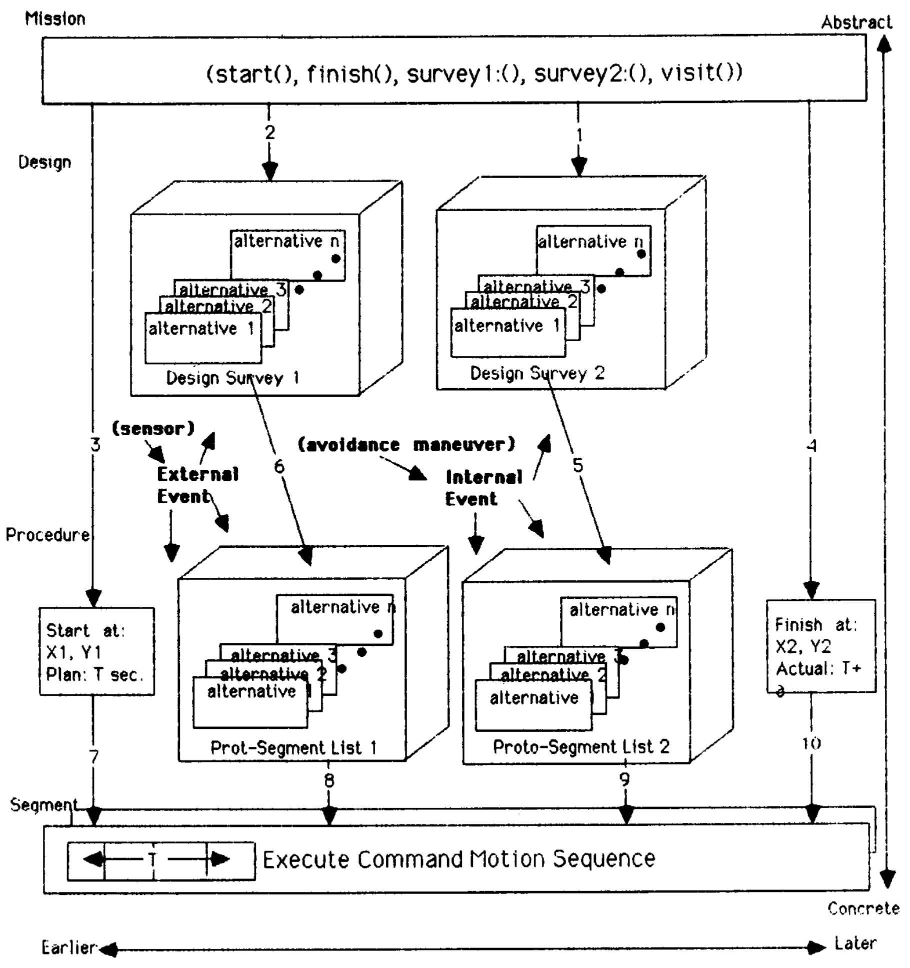


Fig. 22. Problem solving by successive refinement.

compartmentalized, including both the expertise and the operational knowledge, so that object-oriented design tools such as Smalltalk, Flavors, and LOOPS can be used. Further, this allows a newer operating system to be used, wherein programs in any language can be called from the language of the knowledge-based system, so that the overall system is made potentially language-independent.

2) Least Commitment and Alternatives: The concept of multiple blackboards is supplemented by two other features which make this design unique in real-time systems. These are the use of a least commitment strategy and the existence of a reasonable set of alternatives at every level and at all times. The power of this approach is especially clear in the operation of the middle level, which handles analysis and execution for events and processes for which the required response time is on the order of seconds. This level is labeled medium-term response in Fig. 19. The structure and operation of the system at this level will be used as an illustration of these techniques.

Two approaches to planning and execution are shown in Figs. 22 and 23. These are the methods stressed in Hayes-Roth. The first is called the successive refinement method. The second is called temporal/spatial. In either case, execution starts with a description of the mission and an ordering done at the supervisor level and transferred to the medium-term level. The method of illustration adopted is to show how detailed planning for the mission shown in Fig. 20 is effected and modified. The segment names used therefore reflect areas of activity shown in that figure.

Both figures reflect the same basic sequence of transitions in the planning cycle: mission–design–procedure–segment. The first of these is the current projection of the sequence of execution of discrete packets of activity which constitute the components of the mission.

These data are stored and processed at the mission level, where all of the tactics information is stored. The data include maneuvers for known features of the terrain. In effect, these data reflect both known facts and assumptions. The assumptions constitute a set of facts subject to retraction, thus giving rise to the truth-maintenance problem typical of expert system operations.

The second level, the design level, is the level at which planning of the sequence of steps to be used for each packet of activity (such as "move to *x*," or "survey *y*") is accomplished. The survey packets make good illustrations, because

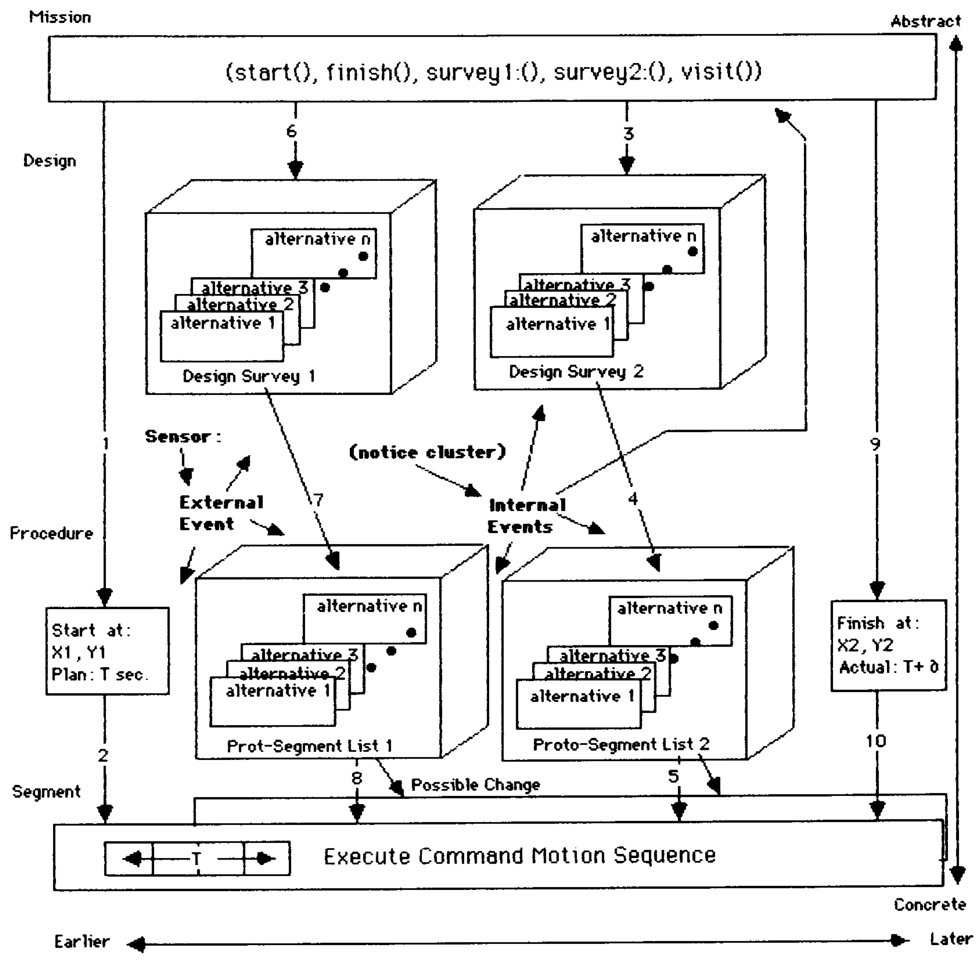


Fig. 23. Problem solving—spatial/temporal ordering.

they consist of specific combinations of linked segments which cover the desired area with a resolution dependent on the amount of detail desired.

The procedure level generates the sequence of motion commands required. Basically, this is the same as the control language of the "turtle" used in Lisp and Logo to introduce children to the concept of programming: move $x1$, turn y deg., move $x2$, etc., until it is supplemented by the added sophistication needed for maneuvers, as required by the environment and the desired physics of the vehicle motion (speed, turn rate, etc.).

The "lowest" level of activity, the segment level, represents execution in "logical" time segments, where a logical time segment is defined as the time duration of a steady-state of activity (forward motion, for example).

Thus far, the design is two-dimensional, as are other current blackboard systems. The basic dimensions are

- vertical: level of abstraction: concrete at the bottom, and abstract at the top;
- horizontal: time of execution: current time at the right, later time at the left.

The most significant new feature of this part of the design, however, is the inclusion of specificially stated alternatives which are active at all times. This is represented as a third dimension, directed into the page of the drawing. It transforms the blackboard design to a design which can encompass three or more dimensions. At the "design" level, the alternatives include such things as

- the ability to change the design of a survey pattern to avoid an obstacle (or to examine in detail an object of interest);
- the ability to adapt the design to variation, such as adding a task, is needed to comply with a change order, such as adding a visit to a cluster, caused by the "notice cluster" event shown in Fig. 23.

The "notice cluster" event is the kind of activity used by Hayes-Roth to demonstrate the utility of the multiple-blackboard control architecture.

Alternatives are also stored at the procedure level, as indicated. The alternatives defined at this area, however, reflect mostly the kind of changes precipitated by the occurrence of external events. Typically, they would consist of revising the instructions for the current segment to provide for the incorporation of an avoidance maneuver. However, activity may also be triggered by a command to interrupt or change a path because of a modified pattern that the mission or design level has created.

The segment path, when executed, does not create any design modifications itself. It is solely concerned with recording or forwarding the specific control signals needed in the current execution frame. However, it has available to it reflex responses, and these can cause switching to an alternative plan, as indicated by the alternate line of execution shown in the background. At this level, the system incorporates "canned" responses to various kinds of object or obstacle which might be encountered. Initially, this consists of the shape classifiers "edge" for a narrow object, "block" for an object of larger extent (but with visible edges), "wall" for an extended object, and "slope" for a smooth transition in shape. Each of these objects carries with it a definition of the characteristics of the sensor information which indicate its presence, and a set of avoidance commands. In the case of the "wall" object, there is an alternative which consists of pausing to collect more data (perhaps including retreat and/or initiating a search for a top or an edge).

The sources of the events which trigger use of the alternatives are sensors for external events, and processing of sensor and state data, for external events. In both cases, the arrows indicate that the occurrence of the event causes activity to propagate through all levels. As indicated by the discussion of the avoidance maneuver, this activity will encompass immediate action while replanning is taking place. The mechanism for recording all of this activity is the set of blackboards or communication media linking knowledge sources at all levels, as well as all knowledge sources at a given level. The interaction of levels of abstraction is indicated by the fact that the "notice cluster" action occurs at the level, but the design activity is affected by it, via a notice procedure.

This linkage illustrates the value of the least commitment strategy. In the event that the new activity should logically proceed before the next survey pattern starts, the design level simply notifies the procedure level of the postponement, and inserts the new activity ahead of it. Since no command strings have been loaded, the impact of the change is minimal.

This mode of operation allows the system to act in an opportunistic way, without the inefficiency of having to discard instructions already generated. The initial mission specification, divided into the logical groups on which segments are based, constitutes a wish list, and an ordering which ensures completion of the mission with a reasonable reserve for fuel and pattern power. It includes a set of additional tasks which may be needed (for example, if specific actions are desired on encountering specific objects) and enough of a model of the environment to cover typical obstacles or terrain features which may require immediate response, with the appropriate response also specified. Fortuitous events, such as the "notice cluster" event, can be included, so that the tasks can be changed to provide more value, if the opportunity arises. The list also includes a list of things "not doable," as well as the "doable" activities in the task descriptions.

The overall operation resembles the operation of a Hypercard stack program on the Macintosh, because the "alternatives" dimension can be thought of as a set of cards which can be selected in any order. Therefore, execution of the mission involves creating a thread through the available sets of cards such that the path may connect to any sequence needed. As a result, the initial estimates of distance, time, and energy are changed, and the actual completion time is changed to the estimated time, plus or minus delta, as indicated.

Minimizing the commitment to detailed plans, combined with making available alternatives at every level, also minimizes the problem of truth maintenance. In the event that certain assumptions must be changed as a result of encounters in the environment, retracing the steps already taken is simplified. Parts of the path may be retraced and redone because dependencies are stored, and the alternatives are known. This operation is the same as the automatic truth maintenance systems of KEE, ART, and Nexpert. In general, however, there will be a limit on the amount of retraction and revision allowed, using the heuristic of "point of no return," assuming that any segment successfully completed need not be retraced, so that the data for it can be discarded. This also minimizes the processing required to modify the set of facts describing the environment, and hence the processing associated with truth maintenance.

The numbers shown in Figs. 22 and 23 indicate the order in which the tasks would be executed, as a function of the scheduling algorithm used. The numbers in Fig. 22 are the order resulting from the use of the successive refinement algorithm described in Hayes-Roth, and the numbers in Fig. 23 result from use of the temporal/spatial algorithm. The successive refinement algorithm favors the results generated at the highest level of abstraction, with the least variability to account for the real world. The spatial/temporal algorithm favors the segment level, where the execution is actually occurring, and where the impact of the real world is significant and immediate. The system uses both algorithms, in the roles appropriate to them, but with the possibility to switch between them. That is, the successive refinement algorithm is the basic method used at the supervisor level, and the spatial/temporal algorithm is best used at the lower levels. In this approach, task designs are not necessarily completed or executed in the order used by the initial program created by the supervisor module, but in the order determined by the temporal/spatial algorithm. Further, segment design is not finalized until the preceding task has been initiated, so that the actual procedure can reflect variations caused by intervening events. This allows the least possible commitment to be made prior to actual need. The system will continue to use a mixture of the algorithms, to allow for alternation between them whenever possible.

The heuristic which is used to cause switching between the algorithms is based on the time available to complete the plan. The allowable planning time is a part of the mission specification, which allows the planning process to be made context-sensitive, and to interact with response-time requirements. It allows the system to adjust both the path and the planning activity to account for the nature of the problem and the capability of the vehicle. If the planning time is greater than a specified limit, then the successive refinement strategy is used, to obtain a more nearly optimum solution. If it is less than the limit, then the spatial/temporal algorithm is used. The difference is that, in the spatial/temporal algorithm, knowledge sources are dispatched in the order in which they request control, so there is no guarantee that the final plan will in any way be optimum.

The results of an execution using each of these algorithms, in steps of operation of the inference engine, are shown in Figs. 24 and 25. These figures show the activity occurring on the execution blackboard. The gaps shown result from activity on the control blackboard, which is not illustrated. Fig. 24 shows results using the successive refinement solution, and Fig. 25 shows results using the spatial/temporal (or "LIFO") solution. The execution to which the figures apply is the mission shown in Fig. 20.

Using the successive refinements technique, there is a general stepping transition from the mission level down to the segment level, with very little stepping back and forth between levels. The mission to procedure to design transitions shown in the "5–6–8" sequence reflect the ability of blackboard control architecture to blend both top-down and bottom-up strategies, as well as the ability of blackboard control architecture to temporarily interrupt the successive refinement strategy.

Using the spatial/temporal technique, the graph has a marked sawtooth shape, reflecting the lack of a control heuristic for focusing the attention of the system at a given level of abstraction until a specified sequence is complete.

Comparing the sequences reveals interesting possibilities. The spatial/temporal strategy produces the first segment, therefore the first motion command, at cycle 18, while the successive refinement strategy does not produce a command until cycle 49. This shows that the successive refinement strategy is not applicable to situations in which the reaction time is short, as in the case of encountering an obstacle. This leads to the concept of having both strategies available: the spatial/temporal for replanning "locally," in the current segment, and the successive refinement for replanning the remainder of the mission, beyond the current segment. This approach also exploits the best features of both approaches. The successive refinement strategy encompasses knowledge of the remainder of the mission as a whole (the "big picture") and therefore can provide judgment on viability. The result is to minimize wastefulness in "backtracking in motion" by maximizing the use of "backtracking in thinking," prior to the issuance of motion commands, as described by Chappell. Since the spatial/temporal technique is subject to the generation of wasteful motion, the mix is appropriate. The heuristics which cause the switching to occur are supplied externally, prior to mission execution. They are constraints which include priorities for the separate parts of the mission, preferred choices, and rules for the manner in which segments are modified.

3) Application Summary: All of these features combine to make EAVE a notable blend of the methodologies of AI to achieve a design which reflects human behavior in a sensible way, but without the pitfalls often associated with attempts to replicate the behavior of an "expert." It is also noteworthy

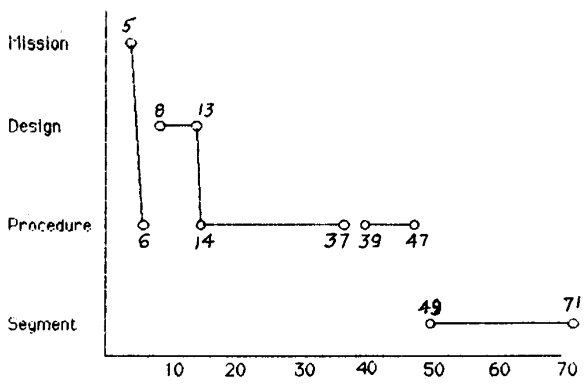


Fig. 24. Successive refinement solution.

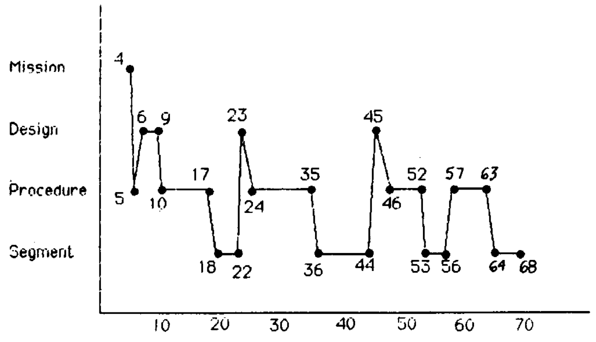


Fig. 25. Spatial/temporal solution.

because it allows for a mixture of technologies, and can easily improve. It is domain-independent, implementation-independent, and architecture-independent, a notable combination of features. A combination of techniques and tools of this kind is found in all of the more successful systems implemented to date.

IX. Research Issues

The major research issues existing in AI today concern the internal structure of the machine used to implement the knowledge-based system (e.g. von Neumann versus neural net) and the relationship between the system architecture and the structure of the problem (e.g. single processor versus multiple processor, or centralized versus blackboard).

The kind of machine used has become an issue because of difficulties in getting the standard machine to effectively mimic important aspects of human behavior. One very important example of a difficult behavior to emulate is the human ability to work with partial information. None of the current approaches, using the von Neumann architecture, provide an adequate solution to this problem. However, an impressive success has been achieved using neural net machines, an emergent technology. It appears likely that the neural net machine will replace both current hardware and current software for realization of expert systems, recognition systems, signal detection and classification systems, and all other systems which involve emulation of the human abilities to detect, identify, and classify, using partial information in a noisy environment. Software is affected because "programming" in the current sense is meaningless for neural net machines—they learn directly from the expert, typically using the form of teaching called "supervised learning" and using an algorithm to control internal organization. The major problem with these systems at present is the lack of tools to allow them to be specified accurately.

The issue of system architecture as compared to the structure of the problem being solved has become an issue because of

- the slowness of execution of programs on standard, centralized machines;
- the independent nature of the various kinds of knowledge and the difference in processing appropriately for the various kinds.

It is clear that, for rule-based systems, "real-time operation," in the sense of military sensors, for example, is not now possible in a single processor. It is also clear that many parts of the knowledge-based system could be executing their operations simultaneously, in a truly parallel process, because they represent different kinds of knowledge. For example, the rules applicable to predicting weather are independent of selection of the type of resources required to prosecute a mission successfully. Therefore, there is much interest in architectures which supply parallel processing, and in a design algorithm which effectively exploits this capability. Candidate system architectures include networks of general-purpose machines, and networks of machines which specialize in performing the same operation on large quantities of data. The first kind of computer is referred to as the distributed computer. The second kind is referred to as the massively parallel computer, a label that is sometimes applied to neural net machines, although the neural net machine is more commonly referred to as the connectionist machine. The major problem on all machines other than the neural net machines has been development of an operating system. The operating system for a distributed system of any kind must be especially concerned with a "reasonable" allocation of multiple resources. Design of a suitable resource allocation algorithm has been a major problem and has been complicated by the inclusion of somewhat arbitrary constraints (such as requiring memory to be shared, and not allowing programs to be passed with data). One simple solution to this problem is to exploit the functional independence of the resident modules. This is particularly attractive because it allows the advantages of the blackboard type of system to be exploited. In the blackboard implementation, however, there is no assurance that each processor will be used to its full capacity at all times. The assurance that the system can provide needed solutions within strict time constraints may, however, override these objections. In addition, the concept of a network of specialists invites the use of processors best fitted to specific parts of the problem, such as neural net machines for classification.

X. Summary

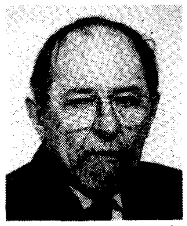
The field of AI has produced tools and methods which have been effective in capturing human expertise and making it more widely available, using traditional machines and specialized Lisp machines. The mixture of techniques now provided in commercial tools allows a more accurate modeling of the nature, the organization, and the operation of human intelligence. This advance has been the result of very large advances in hardware technology, which have drastically reduced the cost of components, especially memory, while simultaneously increasing processing speed. The major problems in using the tools effectively have been a) selection of a suitable problem and b) elicitation of the relevant knowledge. No simple solution exists for the problem of selecting a problem. The best that can be done is summarized in the guidelines given in Section V. As a result of the elicitation problem, it often happens that personnel trained in anthropology, linguistics, and cognitive psychology are more effective at knowledge engineering than personnel trained in computer science. In general, when implementing a system, the more sophisticated tools will be more useful, but some ability to do original programming will be required. To allow for this and also to allow for unexpected difficulty in reproducing the reasoning process, the problem selected should be as simple as possible. At the same time, investment in hardware and software should be minimized as much as possible, so that the rapidly advancing technology can be exploited to the best advantage. This is especially true in view of the new types of machines now appearing in the marketplace. Past successes and emergent technology combined promise the possibility of a totally new generation of systems which will make the machine quite capable of accurately emulating complex human behavior while being built of very small and inexpensive components. The payoff is becoming sufficiently attractive to encourage use of the technology for all kinds of tasks which involve heuristics and human judgment. The practicality of a "knowledge amplifier" is now real, and its availability promises to create a powerful extension of human capability and performance.

References

[1] M. Minsky, "A framework for representing knowledge," MIT for ONR, June 1984 (AD-A011168).
[2] H. J. Berliner, "A prepresentation and some mechanisms for a problem solving chess program," CMU, May 1975 (AD-A012-917/1).
[3] H. J. Berliner, "BKG, A program that plays backgammon," CMU, July 1977 (AD-A043-453/O).
[4] H. J. Berliner, "The B* tree search algorithm, A best first proof procedure," (interim report), CMU, Apr. 1978 (AD-A059-391/3).
[5] "Meta-rules: Reasoning about controls," AL Lab, MIT 56789 (AD-A084639).
[6] "Constructing superview," Dept. of Decision Scienses, Wharton School, U. of Penn. (AD-A093878).
[7] "Natural language communication with machines: An ongoing goal," BBN Rep. No. 5375, July 1983 (an excellent example of the goals described in this report is given by the achievements of Englemann in the TARPS system developed at MITRE Corp.) (AD-A131320).
[8] J. J. Richardson, "Artificial intelligence: An analysis of potential applications to training, performance measurement, and job performance aiding," prepared for the Air Force Human Resources Laboratory, Air Force Systems Command, Brooks AFB, TX., Sept. 1983 (AD-A133592).
[9] Alphatech, Inc., Status report on contract to model the decision-making process of submarine warfare commanders, Feb. 1983 (AD-A133476).
[10] C. Hewitt and P. DeJong, "Analyzing the roles of descriptions and actions in opens systems," MIT AI Lab Rep. AIM 727, Apr. 1983 (essentially a proposal to recreate Smalltalk-80, with object behavior descriptors) (AD-A133614).

[11] R. A. Dillard, "Representation of tactical knowledge shared by expert systems," NOSC TD 632, Oct. 1983 (AD-A136875).
[12] Alphatech, Inc., "Cognitive simulation of anti-submarine warfare commander's tactical decision process," NOSC TR 191, Aug. 1983 (AD-A138849).
[13] C. Hewitt and H. Lieberman, "Design issue in parallel architectures for artificial intelligence," prepared for ARPA, Nov. 1983 (AD-A14282).
[14] D. R. Brown, R. H. Monahan, and W. T. Park, "Artificial intelligence/robotics applications to navy aircraft maintenance," prepared for the David W. Taylor Naval Ship Research and Development Center, Apr. 1983 (AD-A143219).
[15] R. F. Stengel, "Artificial theory and reconfigurable control systems," June 1984 (AD-A143766).
[16] "Some scientific subroutines in LISP," AL Lab, MIT. (AD-A147889).
[17] "The documentation assistant: An intelligent system for documentation," AI&DS, Mountain View, CA (AD-A154709).
[18] "The design and implementation of an object-oriented, production rule interpreter," Naval Postgraduate Schools, 1984 (AD-A154769).
[19] "GOAL: A goal oriented command language for interactive proof construction," Standford AI Lab (not for the faint-hearted; concerned with formal (symbolic) logic and the use of command household tools such as the Takeuchi function and Ramsey's theorem; interesting for sophisticated students of math) (AD-A155093).
[20] NOSC; "Experimental modeling of tactical data systems," TD 548, Oct. 1982 (AD-B070623).
[21] S. M. McFadden, and M. M. Taylor, "Human limitations in towed array sonar recognition," prepared for the Defense and Civil Institute of Environmental Medicine, Ontario, Canada, May 1984 (the first attempt that I have seen to describe the nature of the problem of significant features sensed by humans, so that signal processing operations could be optimized to provide maximum assistance to the human) (AD-B084683).
[22] "A process for the rapid development of systems in support of manager decision-making," AI Lab, MIT (DN83-0045).
[23] "Artificial intelligence and robotics," M. Brady, AI Lab, MIT (DN84-0500).
[24] "Knowledge-based expert systems for off-shore engineering," AI Lab, MIT (DN84-0051).
[25] R. Davis, "Diagnosis via casual reasoning: Paths of interaction and the locality principle," AL Lab, MIT (SN84-0017).
[26] G. Agha, *Actors*. Cambridge, MA: MIT Press, 1986.
[27] I. Asimov, P. S. Warrick, and M. H. Greenberg, Eds., *Machines That Think, Stories about Robots and Computers*. New York: Holt, Reinhardt and Winston, 1983.
[28] C. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[29] H. J. Berliner, "Experiences gained building and testing a chess program," in *Proc. IEEE, Syst. Sci. Cybernetics Conf.* (Pittsburgh, PA), Oct. 14-16, 1970.
[30] H. J. Berliner, "The use of domain-dependent descriptions in tree-searching," in *Perspectives in Computer Science*. London: Academic Press, 39-062, 1977.
[31] H. J. Berliner, "Computer games: The computations of judgement," *Computers and People (USA)*, vol. 30, nos. 3-4, Mar.-Apr. 1981.
[32] H. J. Berliner, "Backgammon computer program beats world champion," *Artificial Intelligence (Netherlands)*, vol. 14, no. 2, Sept. 1980.
[33] H. J. Berliner, "Computer backgammon," *Sci. Amer.*, vol. 242, June 1980.
[34] H. J. Berliner, "Chess," doctoral thesis, CMU, Mar. 1974.
[35] A. Borning, "The programming language aspects of Thinglab, A constraint-oriented simulation laboratory," *ACM Trans. Programming Languages Syst.*, vol. 3, no. 4, Oct. 1981.
[36] A. Borning, "Thinglab—A Constraint-oriented simulation laboratory," Ph.D. dissertation, Stanford University, in 1979; available from University Microfilms International, Ann Arbor, MI.
[37] R. J. Brackhaman and H. J. Levesque, Eds., *Readings*. Los Altos, CA: Morgain Kaufmann Publishers, 1985 (not for the novice, but it gets more readable after the first two papers).
[38] E. C. Carterette and M. P. Friedman, *Handbook of Perception*, vol. IV, *Hearing*. New York: Academic Press, 1978.
[39] E. C. Carterette and M. P. Friedman, *Handbook of Perception*, vol. VII, *Language and Speech*. New York: Academic Press, 1976.
[40] W. L. Chafe, *Meaning and the Structure of Language*. Chicago: Univ. of Chicago Press, 1970.
[41] S. G. Chappell, "A blackboard system for context sensitive mission planning in an autonomous vehicle," presented at 5th Int. Symp. Unmanned, Untetethered Submersible Technol., June 22-24, 1987.
[42] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*. Reading, MA: Addison-Wesley, 1985.
[43] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*. New York: Springer-Verlag, 1981.
[44] R. Davis and D. B. Lenat, *Knowledge-Based Systems in Artificial Intelligence*. New York: McGraw-Hill, 1982.
[45] S. R. Delaney, *Babel-17*. New York: Ace Books, 1966.
[46] A. Elithorn and R. Banerji, *Artificial and Human Intelligence*. North Lolland, 1984 (on order).
[47] P. A. Eschholz, A. Rossa, and V. P. Clark, *Language Awareness*. New York: St. Martin's Press, 1974 (no deep theory, just a set of interesting examples, and fun to read).
[48] E. A. Esper, *Analogy and Association in Linguistics and Psychology*. Athens, GA: University of Georgia Press, 1973.
[49] E. A. Feigenbaum and P. McCorduck, *The Fifth Generation*. Reading, MA: Addison-Wesley, 1983.
[50] E. A. Feigenbaum and A. Bar, *The Handbook of Artificial Intelligence*, vol. I. Stanford, CA: Heuristic Press. Copyright by Kaufman Press, Los Altos, CA; published by William Kaufmann, Inc., 1982.
[51] E. A. Feigenbaum, and P. R. Cohen, *The Handbook of Artificial Intelligence*, vol. III. Stanford, CA: Heuristic Press. Copyright by Kaufmann Press, Los Altos, CA; published by William Kaufmann, Inc., 1982.
[52] J. A. Fishman, *Sociolinguistics, A Brief Introduction*. Rowley, MA: Newbury House, 1972.
[53] J. A. Fodor, *The Language of Thought*. New York: Thomas Y. Crowell, 1975 (an attempt to describe language in the "procedure" terms of a computer, so that it can be created by a machine).
[54] A. Goldberg and D. Robson, *Smalltalk-80, The Language and Its Implementation*. Reading, MA: Addison-Wesley, 1983.
[55] A. Goldberg, *Smalltalk-80, The Interactive Programming Environment*. Reading, MA: Addison-Wesley, 1983.
[56] J. H. Greenberg, *Anthropological Linguistics: An Introduction*. New York: Random House, 1968.
[57] R. K. Guha, "Design of a user microprogramming supporter system," in *Proc. 15th IEEE Computer Soc. Int. Conf.*, 1977.
[58] R. A. Hall, Jr., *Sound and Spelling in English*. Philadelphia: Center for Curriculum Development, 1961.
[59] J. Haugeland, *Artificial Intelligence, The Very Idea*. Cambridge, MA: MIT Press, 1985.
[60] R. Hayes-Roth, D. A. Waterman, and D. B. Lenat, *Building Expert Systems*. Reading, MA: Addison-Wesley, 1983.
[61] B. Hayes-Roth, "A blackboard architecture for control," *Artificial Intelligence*, vol. 26, pp. 251-321, 1985.
[62] P. Henderson, *Functional Programming-Application and Implementation* (Prentice-Hall International Series in Computer Science). Englewood Cliffs, NJ: Prentice-Hall, 1980.
[63] C. F. Hockett, *Language, Mathematics and Linguistics*. The Hague, Netherlands: Mouton & Co., 1962.
[64] R. C. Holt, E. D. Lazowska, G. S. Graham, and M. A. Scott, *Structured Concurrent Programming with Operating Systems Applications*. Reading, MA: Addison-Wesley, 1978.
[65] G. Krasner, *Smalltalk-80, Bits of History, Words of Advice*. Reading, MA: Addison-Wesley, 1983.
[66] C. Levi-Strauss, *The Savage Mind*. Chicago: Univ. of Chicago Press, 1962 (original French edition), 1966 (English edition).
[67] R. E. Levitt, "Using knowledge of construction and project management for automated schedule updating," Stanford and Intellicorp, 1985.
[68] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, no. 2, Apr. 1987.
[69] J. McCarthy, "Recursive functions of symbolic expressions and their computation by machine," *Commun. Ass. Comput. Mach.*, vol. 7, pp. 184-195, Apr. 1960.
[70] W. S. McCullock and W. Pitts, "A logical calculus of the ideas emmanent in neural nets," *Bull. Math. Biophys.*, vol. 5, 1943.
[71] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine Learning, An Artificial Intelligence Approach*. Palo Alto, CA: Tioga Publishing, 1983.
[72] M. Minsky and S. Papert, *Perceptrons, an Introduction to Computations Geometry*. Cambridge, MA: MIT Press, 1969.
[73] M. Minsky, Ed., *Semantic Information Processing*. Cambridge,

MA: MIT Press, 1968 (see especially the thesis by Ross Quillian).
[74] M. Minsky, *Society of Mind.* Cambridge, MA: MIT Press, 1987.
[75] C. V. Negoita, *Expert Systems, Fuzzy Systems.* Menlo Park, CA: Benjamin/Cummings, 1985.
[76] H. P. Nii, "Blackboard systems," *AI Magazine,* vol. 7, nos. 3 and 4, 1986.
[77] D. A. Norman, Ed., *Perspectives on Cognitive Science.* Norwood, NJ: Ablex Publishing, 1980.
[78] D. A. Norman and D. E. Rumelhart, *Explorations in Cognition.* San Francisco: W. H. Freeman & Co., 1975.
[79] J. Piaget and B. Inhelder, *The Psychology of the Child.* New York: Basic Books, 1969.
[80] R. R. Restak, *The Brain.* Toronto, Canada: Bantam Books, 1984.
[81] E. Rich, *Artificial Intelligence.* New York: McGraw-Hill, 1983.
[82] R. C. Schank and P. D. Childers, *The Cognitive Computer.* Reading, MA: Addison-Wesley, 1984.
[83] J. R. Searle, *Speech Acts—An Essay in the Philosophy of Language.* Cambridge, England: Cambridge University Press, 1974.
[84] M. Stefik and G. Brbrow, "Object-oriented programming, Themes and variations," *AI Magazine,* vol. 7, nos. 3 and 4, 1986.
[85] D. S. Touretzky, *LISP, A Gentle Introduction of Symbolic Computation.* New York: Harper & Row, 1984.
[86] D. A. Waterman, *A Guide to Expert Systems.* Reading, MA: Addison-Wesley, 1986.
[87] J. Weizenbaum, *Computer Power and Human Reason.* San Francisco: W. H. Freeman & Co., 1976 (the inside story from the creator of the Eliza program cited in the literature, with cogent comments on some of the "reasoning" programs now available, such as the Macsyma program of Symbolics).
[88] W. E. Welmers, *African Language Structures.* Berkeley, CA: Univ. of California Press, 1973.
[89] H. Whitaker and H. A. Whitaker, Eds., *Studies in Neuro-Linguistics,* vol. 1. New York: Academic Press, 1976.
[90] T. Winograd, *Language as a Cognitive Process.* Reading, MA: Addison-Wesley, 1983.
[91] T. Winograd, *Understanding Natural Language.* New York: Academic Press, 1972 (this is the world of "shrdlu" and the blocks, so often cited in AI literature).
[92] P. H. Winston, *Artificial Intelligence,* 2nd ed. Reading, MA: Addison-Wesley, 1984.
[93] P. H. Winston and K. A. Prendergast, Eds., *The AI Business.* Cambridge, MA: MIT Press, 1984.
[94] P. H. Winston and B. K. P. Horn, *LISP.* Reading, MA: Addison-Wesley, 1984.



Asa B. Simmons received the B.S. degree in physics from the University of Texas (Austin) in 1951 and the M.A. degree in linguistics from California State University (Fullerton) in 1975.

He has had extensive experience in all aspects of system and software engineering, from mission analysis to acceptance tests in the field. His work includes pioneering efforts in system definition and software design for phased-array radars and for trainers. He also has experience in system management, as head of a system design section, overseeing both hardware and software design, and in the field as on-site supervisor for installation and test of air defense ground environment systems for the NATO alliance. He also has extensive experience in system analysis, modeling, and simulation, including event-driven models of systems such as SUBACS. He has received commendations for excellence for software designs he has generated for various sonar trainers. At present, he is occupied with a project to incorporate AI methods in an existing command and control system, with the introduction of neural net technology into company products, and with courses in Lisp, AI, and neural nets. His interests include the nature of intelligence and the methodologies which may be useful in emulating it in machines, where the machines and methods may be of conventional types, or may be based on advanced computation models, as in the case of neural nets.

Mr. Simmons has presented papers in linguistics and has participated in workshops on AI. He also serves on the Corporate Committee for AI. He is a member of the American Association for Artificial Intelligence, the International Neural Net Society, and the Photographic Society of America.

★

Steven G. Chappell received the B.S. degree in botany from the University of New Hampshire, Durham, in 1978. He received the M.S. degree in computer science from UNH in 1987. His thesis was a prototype mission planner for an autonomous vehicle based on a dual blackboard architecture.

He is currently a Software Systems Engineer at the Marine Systems Engineering Laboratory (MSEL) at the UNH, which he joined in 1981. Since then, he has been designing, implementing, and maintaining software for MSEL's experimental autonomous vehicle (EAVE). His interests include autonomous systems, artificial intelligence programming techniques applied to real-world problems, real-time operating systems, parallel processing, and neural networks. He is currently working on a world model for the prototype mission planner and systems software for the third-generation EAVE vehicle.