MEMO A:\AIR&D\TECHPAPR

A PRIMER ON ARTIFICIAL INTELLIGENCE and EXPERT SYSTEMS IN THE PETROLEUM INDUSTRY

E.R.Crain, P.Eng.
D&S Petrophysical, a Division of
D&S Petroleum Consulting Group Ltd
Calgary, Alberta

ABSTRACT

This report is a condensation of the pertinent available literature with respect to artificial intelligence and expert systems which might be of some interest to people within the oil and gas industry. It has been prepared with the view to design and implementation of an expert system for log analysis, based on our proprietory log analysis package, LOG/MATE ESP. The examples used in this report have been phrased in log analysis terms to aid understanding by practioners of this arcane black art.

The report covers the following topics:

No great attempt has been made to be original. Much of the report is condensed from material listed in the Bibliography.

This material, along with the general articles in the Bibliography, will provide sufficient grounding in expert systems terminology for anyone who wishes to become familiar with the subject. Unfortunately, even this condensation is rather lengthy, amounting to some 40 typewritten pages, gleaned and winnowed from over one thousand pages of carefully selected material. Clearly there is much more, including a three volume, 2000 page Handbook of Artificial Intelligence, numerous other major works, plus the transactions of a dozen or more symposia, and the usual plethora of technical journals, published

monthly since 1955.

In the petroleum industry, well log analysis, property evaluation, reservoir simulation, drilling operations, and geologic interpretation have been attacked with AI techniques. Only limited forms of geologic interpretation, log analysis and drilling hydraulics have received any significant attention, however.

The balance of this article provides an overview of expert systems from the petroleum applications point of view, provides definitions of the relevant AI terminology, looks at the tools available for creating expert systems, and reviews progress to date in our industry.

INTRODUCTION

Researchers have worked to develop artificial intelligence for a number of reasons. One is to help understand the human thinking process by modelling it with computers. Another is to make better computer hardware by modelling the computer more closely after the human brain. More achievable goals, such as making computers act more human or easier for humans to use, are also part of the AI spectrum, as are robotics and pattern recognition or artificial vision.

Natural language understanding, automatic translation, and automatic computer programming are other aspects of artificial intelligence.

Until a few years ago, these topics were buried in the academic research environment. Now robots, expert systems for computer configurations and dipmeter analysis, as well as many consultative tasks such as medical diagnostics, are available commercially from the AI community. One pundit once explained that "If it works, it's not AI". This is no longer true.

The distinctions between conventional programming, intelligent programming, and artificial intelligence are not hard and fast. Conventional programming uses procedural languages such as Basic or Fortran to create sequential code to solve explicitly stated problems. Intelligent programming goes one step further. Here data bases are used to hold much of what would otherwise be hard code. As a result, the system is much more flexible, and program sequence or content can be modified at will by the user, as can the knowledge contained in the numeric and algorithmic sections of the data base.

Artificial intelligence software uses a process called symbolic processing instead of linear processing of variables in sequence. Although conventional computing uses symbols (variables) in describing the program, the symbols are not really manipulated by the operating system to create new symbols, relationships, or meanings. In artificial intelligece, new relationships between symbols will be found, if they exist, that were not explicitly stated by the programmer. In addition, symbols without values can be propagated through the relationships until such time as values become available, again without help from the programmer. Anyone who has had a divide by zero error while testing a program will appreciate this feature.

One of the most economically attractive facets of AI is expert systems development. Expert systems apply reasoning and problem solving techniques to

knowledge about a specific problem domain in order to simulate the application of human expertise. Expert systems depend on knowledge about the particular specialty or domain in which they are designed to operate. The knowledge is provided by a human expert during the design and implementation stage, hence the name expert system. Such programs most often operate as an intelligent assistant or advisor to a human user.

The term expert system sometimes has unhappy connotations, such as a computer that is smarter than a human, so the phrase knowledge based system may be used instead. I believe the human ego is strong enough to withstand the label expert system when applied to a computer program.

WHEN AND WHERE ARE EXPERT SYSTEMS USED ?

The uses of expert systems are virtually limitless. The can be used to:

- diagnose
- monitor
- analyze
- interpret
- consult
- plan
- design
- instruct
- explain
- learn
- conceptualize

Thus they are applicable to:

Military and Space Systems

- weapon systems
- target identification
- electronic warfare
- adaptive control
- mission planning
- monitoring
- tracking and control
- communication
- signal analysis
- command and control
- intelligence analysis
- targeting

Industry and Education

- design
- planning
- scheduling
- control
- instruction
- testing
- diagnosis

- monitoring
- maintenance
- repair
- operation

Professions and Consulting (law, medicine, engineering, accounting, law enforcement, software design)

- image analysis
- interpretation
- instruction
- data and text analysis
- specification
- design
- verification
- maintenance
- diagnosis
- treatment

Expert systems are feasible where:

1. there is a high payoff relative to the effort needed to create the system,

2. the problem can only be solved with the help of an expert's knowledge,

3. an expert is available who is willing to formalize his knowledge,

4. the problem may have more than one rational acceptable answer,

5. the problem, solution, and input data descriptions change rapidly over time or space,

6. the problem is never fully defined.

In the petroleum industry, well log analysis, property evaluation, reservoir simulation, drilling operations, and geologic interpretation satisfy the above criteria. Only limited forms of geologic interpretation, log analysis and drilling hydraulics have received any significant attention to date.

WHAT IS AN EXPERT SYSTEM ?

Edward A. Feigenbaum, a pioneer in expert systems, states: "An expert system is an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution. The knowledge necessary to perform at such a level, plus the inference procedures used, can be thought of as a model of the expertise of the best practioners of the field."

Thus, an expert system consists of:

1. a knowledge base of domain facts and heuristics associated with the problem,

2. an inference procedure or control structure for utilizing the knowledge base in the solution of the problem, often called an inference engine,

3. a working memory, or global data base, for keeping track of the problem status, the input data for the particular problem, and the relevant history of what has been done so far.

Figure 1 shows a block diagram of an idealized expert system.

The knowledge in an expert system consists of facts and heuristics. The facts consist of a body of information that is widely shared, publicly available, and generally agreed upon by experts in a field. The heuristics are mostly private, little discussed rules of good judgement that characterize expert-level decision making in the field. The rules may be difficult for the expert to verbalize, and hence are difficult to elicit or share. Some facts and/or heuristics may be proprietory to the user or user's organization, and are thus not shareable outside the organization.

In fact, one of the major uses of expert systems in business is to capture a corporation's overall knowledge base as embodied in the brains of their senior technical and executive staff. The rationale is that the expert system will not retire, get sick, die, or take trade secrets to a competitor.

As an example, the facts in an expert log analysis system are the known properties of rocks and fluids. The heuristics include mathematical rules such as Archie's water saturation equation, as well as usage rules which describe when this equation might be used in achieving the desired results. The inference engine in a conventional log analysis program is the procedural code created by the programmer. It can make only limited, predetermined types of decisions, and cannot reason or show why it took a particular path. An expert system overcomes these drawbacks to conventional programming.

As a prelude to further work on AI in log analysis, these facts and heuristics have been consolidated by the author into a textbook called "The Log Analyst's Handbook", to be published in early 1986 by Pennwell Publishing, Tulsa, OK. The layout and content of the book were specially designed with AI research in mind. However, additional facts, historical fact/result sets, and unspoken heuristics will have to be extracted before a detailed expert system design could be attempted.

When the domain knowledge is stored as production rules, the knowledge base is often referrred to simply as the rule base, and the inference engine as the rule interpreter. It is preferable, when describing real problems, to separate the factual knowledge in the knowledge base into a fact or historical data base, and the heuristics on how to use the facts into a rule base. The two data bases, the rules and the facts, comprise the knowledge base. The reason for this is that facts change rapidly in time and space and heuristics evolve more slowly. Thus some logical separation is desirable. However, this terminology might confuse some AI practitioners, unless these definitions are clearly established.

A human domain expert usually collaborates with a knowledge engineer and a programmer to develop the knowledge base. The synergy between these people is

important to the success of the project. The performance level of an expert system is primarily a function of the size and quality of the knowledge base that it possesses.

It is usual to have a natural language interface to communicate with the user of the system. Menu driven systems are also practical and offer considerable cost advantages, as well as ease of user training. Normally, an explanation module is also included, allowing the user to challenge and examine the reasoning process underlying the system's answers.

An expert system differs from more conventional computer programs in several important respects. In an expert system, there is a clear separation of general knowledge about the problem from the system that uses the knowledge. The rules forming a knowledge base, for example, are quite divorced from information about the current problem and from methods for applying the general knowledge to the problem. In a conventional computer program, knowledge pertinent to the problem and methods for utilizing it are often intermixed, so that it is difficult to change the program. In an expert system, the program itself is only an interpreter and ideally the system can be changed by simply adding or deleting rules in the knowledge base.

Not all modern software is as clumsy to change as suggested above. Our existing LOG/MATE ESP log analysis/geological/engineering workstation is designed along AI lines, by separating the data that drives the plot, print, and compute modules from the interpreters which create the result.

USING AN EXPERT SYSTEM

There are three different ways to use an expert system, in contrast to the single mode (getting answers to problems) characteristic of the more familiar type of computing. These are:

1. **getting answers to problems -- user as client,**

2. **improving or increasing the systems's knowledge -- user as tutor,**

3. **harvesting the knowledge base for human use -- user as pupil.**

Users of an expert system in mode (2) are known as domain specialists or experts. It is not possible to build an expert system without at least one expert in the domain involved in the project.

An expert system can act as the perfect memory, over time, of the knowledge accumulated by many specialists of diverse experience. Hence, it can and does ultimately attain a level of consultant expertise exceeding that of any single one of its "tutors." There are not yet many examples of expert systems whose performance consistently surpasses that of an expert. There are even fewer examples of expert systems that use knowledge from a group of experts and integrate it effectively. However, the promise is there.

To accomplish this task, an expert system must have a method for recognizing and remembering new facts and heuristics while the system is in use, and for gracefully forgetting those which are inconsistent, incorrect, or obsolete. At

the moment, most expert systems require that such changes be made off-line from actual program execution.

THE KNOWLEDGE BASE

Knowledge representation in the knowledge base is an important aspect of expert system design. The three major forms of knowledge representation are production rules, frames, and semantic sets. The different methods are used for different data types and data uses. Production rules are used where IF...THEN statements define the knowledge adequately. Frames are used to represent descriptive and relational data that cluster or that conform to a stereotype. Semantic sets are most useful for defining classifications, physical structures, or causal linkages.

The most popular approach to representing the domain knowledge needed for an expert system is by production rules, also referred to as SITUATION-ACTION rules or IF-THEN rules. Thus, a knowledge base may be made up mostly of rules which are invoked by pattern matching with features of the problem as they currently appear in the global data base. A typical rule for a log analysis system might be:

```
  IF matrix density is greater than sandstone matrix density
 AND lithology is described as shaly sand
THEN suspect a heavy mineral OR cementing agent
  OR suspect inadequate shale corrections
  OR suspect poor log calibrations
```

Most conventional log analysis programs contain checks and balances of this type, coded in Basic or Fortran, with appropriate action being dictated by user defined logic switches. The virtue of an expert system knowledge base is that the expert can modify this rule set with comparative ease, compared to a hard coded program. LOG/MATE ESP contains these rules in a user accessable data base, so the same change can be implemented easily also. In this case, the rule must be formulated mathematically, although the output may be a text message similar to the ACTION part of the rule described above.

The knowledge base may also contain large amounts of quantified data or algorithms which help quantify data. In the petroleum industry, such data may represent the physical and chemical properties of rocks and fluids, or costs and income data for different production environments, or predictive equations which quantify empirical and well accepted rules of thumb. Equations which predict porosity from sonic travel time or production rate from exponential decline are well known examples.

In the petroleum environment, it is inconceivable that an expert system could be successful without extensive information of this type in its knowledge base. Much of our rule base consists of empirical rules of thumb which have been quantified by many experts, and used by larger numbers of practitioners.

This information can be gleaned from literature search, from review of input data, analysis parameters, and comparison of ground truth versus output from prior work, and from manipulation of known data using the laws of physics and chemistry. Thus, a large fraction of the knowledge base does not come directly from the brain of a single expert, but is really a digest of the reference material he would use while performing his analysis. This information is sometimes called world knowledge, but it is still very specific to the domain in question.

Most existing rule-based systems contain hundreds of rules, usually obtained by interviewing experts for weeks or months. In any system, the rules become connected to each other by association linkages to form rule networks. Once assembled, such networks can represent a substantial body of knowledge, although some of it may be incomplete, contradictory, fuzzy, or plain wrong.

In LOG/MATE ESP, we call these networks by the generic label of ROUTINE, which is an assemblage of individual algorithms connected by conditional branching logic. The routine, with its associated computation parameter files and raw data records, constitutes the specific rule network which will be used on this data set. Unfortunately, the network must be created manually, usually by an expert, and tuned for each subsequent use, usually by a low level user with or without the guidance of a human expert.

Although LOG/MATE ESP has an extensive rule base, and can have an extensive knowledge base as well, it is not yet an expert system because it cannot perform any reasoning - it cannot chose the most likely rule network to use for a particular problem. A diagram of the data base for LOG/MATE ESP is shown in Figure 2; it has been especially designed to contain rules, facts, global data, input data, and answers, in anticipation of adding or interfacing an inferencing technique to the system.

An expert usually has many judgemental or empirical rules, for which there is incomplete support from the available evidence. In such cases, one approach is to attach numerical values (certainty factors) to each rule to indicate the degree of certainty associated with that rule. In expert system operation, these certainty values are combined with each other and the certainty of the problem data, to arrive at a certainty value for the final solution. Fuzzy set theory, based on possibilities, can also be utilized.

Often, beliefs are formed or lines of reasoning are developed based on partial or errorful information. When contradictions occur, the incorrect beliefs or lines of reasoning causing the contradictions, and all wrong conclusions resulting from them, must be retracted. To enable this, a data-base record of beliefs and their justifications must be maintained. Using this approach, truth maintenance techniques can exploit redundancies in experimental data to increase system reliability.

THE INFERENCE ENGINE

As indicated earlier, an expert system consists of three major components, a set of rules, a global data base and a rule interpreter. The rules are actuated by patterns, (which match the IF sides of the rules) in the global data base. The application of a rule changes the system status and therefore the data base, enabling some rules and disabling others. The rule interpreter uses a control strategy for finding the enabled rules and deciding which rule to apply. The basic control strategies used may be top down (goal driven), bottom up (data driven), or a combination of the two that uses a relaxation-like convergence process to join these opposite lines of reasoning

together at some intermediate point to yield a problem solution.

The rule interpreter, or control strategy, is often called the problem solving paradigm or model in the AI literature. Other terms used are the inference engine, the solution protocol, reasoning, or deduction.

The essential difference between conventional programming and expert systems is this ability to reason or deduce; to take alternate paths, not based on pre-ordained switches, but based on logical rules and the current state of the global data base.

The problem-solving model, and its methodology, organizes and controls the steps taken to solve the problem. One commonplace but powerful model involves the chaining of IF-THEN rules to form a line of reasoning. If the chaining starts from a set of conditions and moves toward some possible remote conclusion, the method is called forward chaining. An example might be building a custom tailored minicomputer, in which a list of desired features leads to a goal of a complete detailed system configuration parts list. Forward chaining usually is used to construct something.

If the conclusion is known (eg., it is a goal to be achieved), but the path to that conclusion is not known, then working backwards is called for, and the method is called backward chaining. For example, a set of botanical descriptions ought to lead to a species name by backward chaining to find the set of conditions in the knowledge base which match the plant description at hand. Backward chaining methods are usually used for diagnostic purposes; they start from a list of symptoms and attempt to find a cause which would explain the symptoms.

The problem with forward chaining, without appropriate heuristics for pruning, is that you would derive everything possible whether you needed it or not. For instance, the description of a chess game from its possible opening moves creates an enormous explosion of possibilities. If every elementary particle in in the universe were a computer operating at the speed of light, the universe is not old enough to have computed all possible combinations.

Backward chaining works from goals to subgoals The problem here, again without appropriate heuristics for guidance, is the handling of conjunctive subgoals. Conjunctive goals are those which interact with each other, and which must be solved simultaneously. To find a case where all interacting subgoals are satisfied, the search can often result in a combinatorial explosion of possibilities too large for real computers.

Thus appropriatre domain heuristics and suitable inference schemes and architectures must be found for each type of problem to achieve an efficient and effective expert system. There are no universal, general purpose expert systems. Further information on these methods can be found in APPENDIX 1.

The knowledge of a task domain guides the problem-solving steps taken. Sometimes the knowledge is quite abstract; for example, a symbolic model of how things work in the domain. Inference that proceeds from the model's abstractions to more detailed, less abstract statements is called model-driven inference and the problem-solving behavior is termed expectation driven.

Often in problem solving, however you are working upwards from the details or the specific problem data to the higher levels of abstraction, in the

direction of what it all means. Steps in this direction are called data driven. If you choose your next step either on the basis of some new data or on the basis of the last problem-solving step taken, you are responding to events, and the activity is called event driven.

A NOT SO TRIVIAL EXAMPLE

It was not difficult to think of a knowledge base as described earlier. Many computer programs already have them. Humans work easily with tables of data or lists of procedural steps. It is much more difficult to conceive of reasoning or deduction in a computer program, although the simple examples given above suggest the possibilities.

Consider the drawing of the three animals in Figure 3. Humans with prior experience can recognize the difference between them virtually instantly, can name the species and sex, and guess their approximate ages. Some people may even be able to tell the breed of the animals. Could an expert system do the same ?

First, try writing down a list of descriptive features that you know for each of these three animals. Do not rely solely on the characteristics in the drawing. Include enough information so that none of these animals could be mistaken for a zebra or a dog. Then check off on each of your lists the observable features of each animal in the illustration. Does your checklist identify each animal uniquely ? Keep improving your list until there is no doubt. You may need a number of conditional statements, using "AND" and "OR" to make identification positive, or even some numerical procedures or probabilities to handle extreme cases.

We have just described the process of extracting knowledge from an expert and using inferencing to draw conclusions. Backward chaining in an expert system would check the checklists, and a reasonable pattern match would generate an answer as to the animal's species, along with a statement as to its probable chance of being correct.

In this case, to emulate the human brain's ability to do pattern recognition, we had to resort to a brute force listing of pattern features, a semi-quantitative description of the animals. Various heuristics would be needed in a real program to account for the fact that you cannot "see" all around the animal in a drawing, and must make assumptions about symmetry and hidden features. After all, this may only be a drawing of a drawing of an animal, and not a real animal at all,

Now try the animal in Figure 4 on your checklists. Did you identify the animal right away or did you need further updates to your knowledge base? Did any of your updates create conflicts or contradictions? This process describes the "expert as tutor" mode of operation.

Expert systems are not good at pattern recognition from outline drawings such as these, but do better on quantized lists of facts and relationships as described in our example. Real pattern recognition is coming - especially in military and aerospace applications such as target identification and response strategies.

To complete this exercise, consider the possibility of having more data, such as X-rays of the animals' skeleton, autopsy and dissection results. or even a drawing or photograph of other views of the animal. This information would make identification much easier, and allow the programmer to create many new rules, and to add to the factual data base.

These sets of extra data are analogous to extra well logs or extra non-log data, such as core, test, and production history information. Obviously, with more facts to work on, and more rules to evaluate, an expert system to determine animal species or the production to be expected from a well, will do a better job. Thus integration of various disciplines in a common knowledge base is a natural outcome of expert system research.

PROBLEM SOLVING TECHNIQUES

Different types of experts use different approaches to problem solving. Knowledge, for example, can be represented in many different ways. Similarly, there are many different approaches to inference and many differnt ways to order one's activities (See Appendix 1). Generalized models (paradigms) are available in the form of system building tools.

A consultation paradigm is a generic conception of a particular type of problem solving that is common to several different domains. Thus, we refer to one consultation paradigm as the diagnosis/prescription paradigm. The name derives from medical problems, such as diagnosing infections and recommending drugs. Many other medical problems also seem to involve a similar approach to problem solving. But problems in various nonmedical situations often seem to require similar expertise; reviewing a set of symptoms, considering various possibilities, and then recommending actions based on a qualified estimate of the probable causes. Most petroleum related expert systems use some form of consultative model.

LANGUAGES AND TOOLS

Tools allow knowledge engineers to construct knowledge systems to help users solve problems that can be described in terms of one, or at most, a very few consultation paradigms.

Knowledge representation, inference, and control strategies are specific software techniques. In some cases one technique, such as certainty factors, will contribute to a solution for more than one consultation paradigm. On the other hand, some techniques are strongly associated with particular paradigms. In general, specific types of problems imply tools that are built up with a certain set of representation, inference, and control techniques.

There is not, however, a one-to-one match between software techniques and problems. One programmer may approach a constraint satisfaction problem using a tool based on backward chaining; another knowledge engineer, faced with the same problem, might choose a tool that relies on forward chaining. However, few knowledge engineers would probably choose to use a backward chaining tool

to tackle a complex planning problem, because it is known to be an inappropriate model.

When choosing a tool, you want to be very sure that the specific tool choosen is appropriate for the type of problem on which it is to be used. Unfortunately, since knowledge engineers do not understand how to handle most of the problems that human experts routinely solve, and since there are only a few tools available, many types of expert behavior cannot be conveniently encoded with any existing tool.

Thus in most cases, managers who want to employ knowledge engineering techniques have a choice. They can focus on problems that are well understood and ignore those for which there are no available solutions at this time. Or they can develop a sophisticated knowledge engineering team and try to build a system by creating a unique set of knowledge representation, inference, and control techniques in some general-purpose AI language or environment such as INTERLISP, PROLOG, or perhaps OPS5. This is clearly too expensive for most small to medium sized companies.

Most companies have decided to focus on solving problems for which there are already established tools. Given the large number of available problems with significant paybacks, this is certainly a reasonable strategy. Moreover, even companies that have decided to develop a team capable of creating unique knowledge systems have usually built that team while working on some fairly well-understood problem.

The tools used by the expert system community involve specialized computer languages and system building tools, as well as specialized hardware architecture, often called LISP machines after the dominant language used in the USA. The other popular language, used mostly in Europe and Japan, is Prolog. Other languages are used in limited areas. The specialized hardware is not described further in this paper.

The conventional languages, such as Basic and Fortran and many others, have been successfully used to create expert systems. The AI community tends to downplay these successes, and insist on using LISP. It should be remembered that LISP was invented at a time when Fortran could not handle strings of characters at all. Much invention has since taken place and extended Basic and other languages handle user defined functions, recursion, and text strings quite well, all deficiencies which LISP was supposed to overcome. LISP is also very difficult to read, and programmers often cannot understand or debug each others code, in contrast with structured extended Basic which can be composed so as to read well in pseudo English.

In addition to the true languages, the system building tools can be divided into three groups:

1. Small system building tools that can be run on personal computers. These tools are generally designed to facilitate the development of systems containing less than 400 rules and are not discussed further here.

2. Large, narrow system building tools that run on LISP machines or larger computers and are designed to build systems that contain 500 to several thousand rules but are constrained to one general consultation paradigm.

3. Large, hybrid system building tools that run on LISP machines or larger

computers and are designed to build systems that contain 500 to several thousand rules and can include the features of several different consultation paradigms.

Details of the program languages commonly used for expert system development, and some of the expert system development environments available commercially, are described in APPENDIX 2 and listed by type in Table 1.

PETROLEUM INDUSTRY EXAMPLES

The following material is taken from various references, listed in the Bibliography. It describes the best known petroleum applications in considerable detail, so as to provide a starting point for discussion and planning for an expert system for log analysis.

The examples described are:

1. Drilling Advisor	Elf-Aquitaine	Figure 5
2. Prospector	Stanford	Figure 6
3. Dipmeter Advisor	Schlumberger	Figure 7
4. Expert Log Analysis System	Amoco	Figure 8
5. Mudman	Baroid	No illustration

These systems demonstrate a variety of methods and implementation techniques.

DRILLING ADVISOR

DRILLING ADVISOR is a prototype knowledge system developed for the French oil company Societe Nationale Elf-Aquitaine (ELF) by Teknowledge Inc. The system is designed to assist oil rig supervisors in resolving and subsequently avoiding problem situations. The oil rig supervisor is familiar with the technology, equipment, and procedures involved in the drilling process, but occasionally requires assistance when special problems occur.

Normally, an expert is flown to the rig site when such problems occur. Since it is not unusual for drilling-related expenses to exceed $100,000 per day or for shutdowns related to special problems to last for several weeks until an expert can be brought to the site, the savings that an on-rig knowledge system could effect are considerable.

Teknowledge and Elf agreed to develop a prototype system to solve one specific problem, down-hole sticking, which occurs when the rotary and vertical motion of the drill is impeded.

DRILLING ADVISOR was developed by means of a tool called KS300, an EMYCIN-like tool. Thus, DRILLING ADVISOR is a backward chaining, production rule system, like MYCIN, that takes full advantage of EMYCIN's user-friendly interface and knowledge aquisition facilities.

By using KS300, Teknowledge was able to develop the initial problem assessment and design in a little under three months and was able to develop a prototype of the drilling advisor sticking system in a little under nine months.

DRILLING ADVISOR has been implemented on two different systems. It can be run on either a DEC 20 or Xerox 1100 machine.

Currently the knowledge base of DRILLIING ADVISOR consists of some 250 rules. Approximately 175 of those rules are used in diagnosis, and the other 75 rules are used in prescribing treatment. Results to date are very encouraging. The system has successfully handled a number of difficult cases that were not included in the set used during its development. Current plans call for extending the capabilities of DRILLING ADVISOR and for integrating it into the actual drilling environment.

PROSPECTOR

PROSPECTOR has one foot in the world of research and the other in the world of commercial applications. It was developed in the late 1970's at Stanford Research Institute (SRI) by a team that included Peter Hart, Richard Duda, Rene Reboh, K. Konolige, P. Barrett, and M. Einandi. The development of PROSPECTOR was funded by the U.S. Geological Survey and by the National Science Foundation.

PROSPECTOR is designed to provide consultation to geologists in the early stages of investigating a site for ore-grade deposits. Data are primarily surface geological observations and are assumed to be uncertain and incomplete. The program alerts users to possible interpretations and identifies additional observations that would be valuable to reach a more definite conclusion.

PROSPECTOR is, broadly speaking, a descendant of MYCIN, but it was not developed using the EMYCIN system building tool. In fact, PROSPECTOR has resulted in a new tool, called KAS. PROSPECTOR goes beyond MYCIN in a number of important ways. The knowledge base of PROSPECTOR, for example, is based on a semantic network organized, in turn, around five different geological models. Each model describes the information and relationships that pertain to a particular type of mineral deposit. The PROSPECTOR team worked with different mineral experts to develop the different models.

In effect, assertions are nodes in the network. Typical assertions include:

> **"There is pervasively biotized hornblende."**
>
> **"There is alteration favorable for the potassic zone of a porphyry copper deposit."**

Each assertion is either unknown, true, false, or assumed to be true and assigned some probability. The arcs connecting nodes of the networks are inference rules. Each rule specifies how the probability of one assertion will affect the probability of another rule. In effect, PROSPECTOR's inference rules are the same as MYCIN's production rules. Additional inference rules are used to establish assertions and to order search.

PROSPECTOR is much more flexible than MYCIN when it interfaces with users. To begin with, it employs a constrained natural language interface (LIFER) that allows the user to type sentences just as they would ask questions of a

geological consultant. LIFER interprets the sentences for PROSPECTOR.

In addition, PROSPECTOR is a "mixed-initiative" system. The user can volunteer information whenever he or she wishes. Thus, one of the major user complaints about MYCIN is eliminated. The user can begin a session by telling PROSPECTOR everything known. The user can stop PROSPECTOR whenever desired and provide additional information. PROSPECTOR immediately inserts the volunteered data into its inference network and adjusts its strategies and questions accordingly. The basic control strategy, once the user stops volunteering information, is backward chaining.

PROSPECTOR can also accept input in the form of raw data and generate a graphic response. Thus, the user can enter informaiton about a site and PROSPECTOR can generate a new map showing conclusions about the site.

Once the user has volunteered initial data, PROSPECTOR inserts the data into its models and decides which model best explains the given data. Further confirmation of that model then becomes the primary goal of the system, and the system asks the user questions to establish the model that will best explain the data. If subsequent data cause the probabilities to shift, of course, the system changes priorities and seeks to confirm whichever model seems most likely in light of the additional data.

In 1980, as a test, PROSPECTOR was given geological, geophysical, and geochemical information supplied by a group that had terminated exploration of a site at Mt. Tolman in Washington in 1978. PROSPECTOR analyzed that data and suggested that a previously unexplored portion of the site probably contained an ore-grade porphyry molybdenum deposit. Subsequent exploratory drilling has confirmed the deposit and, thus, PROSPECTOR has become the first knowledge-based system to achieve a major commercial success. The weakest part of PROSPECTOR's performance was its failure to recognize the full extent of the deposit it identified.

PROSPECTOR's five models represent only a fraction of the knowledge that would be required of a comprehensive consultant system for exploratory geology. SRI continues to develop and study PROSPECTOR, but there are no plans to market the system. The principal scientists who developed PROSPECTOR and KAS, the expert system building tool derived from PROSPECTOR, have left SRI to form a private company (Syntelligence) and Ms. Reboh has taken a position at the University of Calgary.

Thus, PROSPECTOR, like MYCIN, has never become an operational system. Its innovations and successes, however, have inspired a large number of knowledge engineers, and there are a number of commercial systems under development that rely on one or more of the features first developed and tested during the PROSPECTOR project.

THE DIPMETER ADVISOR

Unlike fanciful movie images, oil is rarely discovered in gushers that send it spewing out of the ground. More typically, the discovery and draining of fields is a painstaking process involving inferred reconstruction of underground geology. The presence of prehistoric beaches, deltas, and faults several thousand feet underground are important information suggesting the likely location of oil-bearing formations.

The reconstruction process is based in large part on measurements provided by a number of probes called well logs. The probes are lowered into a well and then slowly retrieved, measuring various physical properties of the rock every few inches as they ascend, Since a log may be as much as 10,000' long, and may make many simultaneous measurements, there is a significant amount of data to be interpreted.

One of the important and widely used probes is the dipmeter, which yields information about the orientation of rock layers. From its measurements the inclination, or dip, of the subsurface can be computed. Other commonly used logs provide measurements from which such properties as rock resistivity and porosity can be determined.

Interpretation of a dipmeter and related logs requires inferring the presence of large-scale, three dimensional geologic formations from small-scale, two-dimensional information about physical properties.

The task is suited to the expert systems paradigm for several reasons. First, there are recognized human experts who routinely solve the problem, providing both an acknowledged source of expertise that can be tapped to help build the knowledge base and a standard by which to judge program performance. Second, skill at this task is acquired via training and experience. Becoming an interpreter involves explicit study and the skill is in large measure cognitive, rather than perceptual. Both of these make it more likely that it can be captured as a collection of inference steps.

Finally, the domain is at the appropriate stage of development. It is sufficiently well established that it has a vocabulary of basic concepts and a collection of informal but useful rules of thumb, but is not yet so well developed that there is a uniform and reliable general solution method. At this stage of development a qualitative, symbolic reasoning approach can be very effective.

Work on this task also has a strong pragmatic motivation. The field of log interpretation is at present manpower-limited. Given the current emphasis on exploration, a program capable of high performance on this task would have considerable utility.

The Dipmeter Advisor system attempts to emulate human expert performance in dipmeter interpretation. It utilizes dipmeter patterns together with local geological knowledge and measurements from other logs. It is characteristic of the class of programs that deal with what has come to be known as signal to symbol transformation. The program is written in INTERLISP and operates on the Xerox 1100 Scientific Information Processor (Dolphin).

The system is made up of four central components:

(i) a number of production rules partitioned into several distinct sets according to function (eg., structural rules vs stratigraphic rules)

(ii) an inference engine that applies rules in a forward-chained manner, resolving conflicts by rule order

(iii) a set of feature detection algorithms that examines both dipmeter and open hole data (eg., to detect tadpole patterns and identify lithological

zones)

(iv) a menu-driven graphical user interface that provides smooth scrolling of log data.

Conclusions are stored as instances of one of 65 token types, with approximately 5 features/token, on a blackboard that is partitioned into 15 layers of abstraction (eg., patterns, lithology, stratigraphic features). There are 90 rules and the rule language uses approximately 30 predicates and functions. The rules have the familiar empirical sssociation flavor. A sample is shown below. This sample is similar to the actual interpretation rule, but has been simplified somewhat for presentation.

```
   IF  there exists a delta dominated, continental shelf marine zone
   AND there exists a sand zone intersecting the marine zone
   AND there exists a blue pattern within the intersection

   THEN  assert a distributary fan zone

   WITH        top = top of blue pattern
   WITH        bottom = bottom blue pattern
   WITH        flow = azimuth of blue pattern
```

The system divides the task of dipmeter interpretation into 11 successive phases as shown below. After the system completes its analysis for a phase, it engages the human interpreter in an interactive dialogue. He can examine, delete, or modify conclusions reached by the system. He can also add his own conclusions. In addition, he can revert to earlier phases of the analysis to refer to the conclusions, or ot rerun the computation.

1. Initial Examination: The human interpreter can peruse the available data and select logs for display.

2. Validity Check: The system examines the logs for evidence of tool malfunction or incorrect processing.

3. Green Pattern Detection: The system identifies zpnes in which the tadpoles have similar magnitude and azimuth.

4. Structural Dip Analysis: The system merges and filters green patterns to determine zones of constant structural dip.

+5. Prelimanary Structural Analysis: The system applies a set of rules to identify structural features (eg., faults).

6. Structural Pattern Detection: The system examines the dipmeter data for red and blue patterns in the vicinity of structural features. The algorithms used by the system to detect dip patterns are beyond the scope of this paper. It is worth noting, however, that textbook definitions do not provide sufficient specification. The problem is complicatd by local dip variations and occasional gaps in the data.

+7. Final Structural Analysis: The system applies a set of rules that combines information from previous phases to refine its conclusions about structural features (eg., strike of faults).

8. Lithology Analysis: The system examines the open hole data (eg., gamma ray) to determine zones of constant lithology (eg., sand and shale).

+9. Depositional Environment Analysis: The system applies a set of rules that draws conclusions about the depositional environment. For example, if told by the human interpreter that the depositional environment is marine, the system attempts to infer the water depth at the time of deposition.

10. Stratigraphic Pattern Detection: The system examines the dipmeter data for red, blue, and green patterns in zones of known depositional environment.

+11. Stratigraphic Analysis: The system applies a set of rules that uses information from previous phases to draw conslusions about stratigraphic features (eg., channels, fans, bars).

For the phases shown above, "+" indicates that the phase uses production rules written on the basis of interactions with an expert interpreter. The remaining phases do not use rules. The rules obtained to date are due to J.A. Gilreath of Schlumberger Offshore Services, New Orleans, LA. The feature detectors and signal processing algorithms were written independently by project members. The scrolling graphics code was written by Paul Barth. Extensions to the INTERLISP-D menu package were written by Eric Schoen.

ELAS: Expert Log Analysis System

ELAS is an expert system front end for Amoco's interactive log analysis package, which runs on an IBM mainframe-terminal configuration. The front end was written with the EXPERT tool, and is used to prompt a user through the log analysis steps of the interactive program.

This form of expert system is often called a surface level model. The surface level model is of the production rule type, whereas the deep model is of purely mathematical description, expressed as a set of equations. The latter are implemented as complex software tools, such as reservoir simulators or log analysis packages.

From a practical applications point of view, well log interpretation represents an important problem, since it permits an assessment of the likely presence of hydrocarbons and possible yields of the well during exploration and production. From the perspective of expert systems research, this application is proving very helpful in increasing the understanding of representation, communication, and control processes in multi-level systems. And, from the more general software engineering point-of-view, we are learning how one might exploit existing software systems more fully by building a coordinating and advisory system that makes these programs easier to use by a wider variety of expert and non-expert users alike.

In many problem areas, it is not unusual to find that valuable software has already been developed to aid the expert in data analysis, the design of experiments, and the interpretation of results. These programs are often quite complex packages, developed over several years and enhanced through extensive user experience. In designing an expert system, it is only natural that one should want to take advantage of such software.

One of the first efforts in modeling expert advice on the use of a complex

program was the SACON project which developed an advisory model for the MARC structural analysis program. However, there was no interaction between the two programs. SACON was run before the MARC program, giving advice on its prospective use. In order to develop an expert system to its fullest potential, interaction is needed between the advising program and the application programs.

In a sophisticated system, the interpretive program will be fully integrated with the application programs, so that they communicate their results to one another, and advice changes dynamically as the model tracks the user interaction. Furthermore, the system must have the ability to automatically take a recommended action if the user agrees. In effect, we will have a program that not only gives advice, but also can accept the advice and act on it.

A rule based advice model, called ELAS, has been integrated with the existing Amoco software for well-log analysis. In order to accomplish this task, original well-log software was reorganized so that its use could be monitored and controlled. Its representation was structured according to the types and sequences of methods used by expert analysts. By allowing the user to vary the assumptions and parameters used in different individual analyses, the system makes available interactive interpretations of the alternative approaches that an expert might take to a complex problem of well-log analysis.

ELAS runs on a variety of systems (including IBM's VM/CMS and DEC's VAX/VMS) using a dual terminal configuration: a graphics and an alphanumeric terminal with a shared keyboard. The system allows the user to interactively perform experiments in the analysis of logs. Advice is generated based on the results of previous experiments, and a running summary is kept of the actions already taken. The advisory system is based on the EXPERT rule scheme.

To make interaction easy for users, the front end of the ELAS system has, at its top level, a master panel which holds a snap-shot of the current status of the analysis of an already selected well. The columns correspond to different geological zones (by depth) that have been chosen for analysis. The rows correspond to the parameters for the zones. Initial values for some of the parameters must be supplied by the user. Many of them, however, may be obtained through subsequent analysis. Some parameters may stand for specific tasks that the user might want to invoke to help in the analysis.

Most user-program communication is controlled through this master panel. It is displayed on the graphics screen and includes a concise set of key parameters and tasks that are crucial in well-log analysis. A parameter may be aconstant, a log (represented as a vector of digitized values for each foot of depth in the well), or an expected characteristic of the well (eg., the presence of gas in some zone).

There is a superficial similarity to a spreadsheet program, the concept used by many personal computer programs. In the simpler environment of a spreadsheet, we see a program that presents information in a concise format and allows the user to vary a parameter and then watch all dependent results change. ELAS is faced with a much more complicated computational task, but it tries to show the propagation of effects that follow from the user's change of a parameter value or choice of analysis method within as short a time as possible ranging from almost instantaneous to many seconds. This is accomplished by updating the master panel, after which the user may invoke more detailed panels or displays for the specific methods. Changing a parameter may imply quite a large number of computational steps and not all information can be described in a simple tabular format.

Upon request, ELAS can provide interpretations and recommendations to the user. The advice is organized along the topics indicated on the master panel. While the master panel appears on the graphics terminal, the advice is always given on the standard terminal so that they may be viewed simultaneously.

ELAS allows the user to direct both the mathematical analysis and the interpretive analysis by changing parameters or invoking tasks through the master panel. The outcomes of mathematical analyses that follow are then reported back to the user through this same panel. The expert system updates its interpretive analysis after every change in the evidence so that it always reflects the current status of the panel. The system also synchronizes all derived logs that are affected by changes in the panel parameters or methods. Changes are made either through user action or updates in the mathematical analysis. The user has the freedon to carry out an entire well-log analysis sequence without ever asking for advice from the system, or advice may be requested at any stage of the analysis.

To illustrate how different but related mathematical methods of log analysis are integrated into ELAS, a simplified example of a formula used very frequently (Archie's equation) is described here. This equation is used for computing water saturation in a zone of interest. This calculation is important because once a zone is identified as bearing hydrocarbons, the fluid present in that zone is a combination of water, oil, and gas. If the amount of water is known, we can therefore find, by subtraction, the fractions of other fluids, which would be oil and gas. The parameters in this equation vary depending on the kind of lithologic formations downhole.

The variables of this equation are quantities that can be changed through the master panel. A change in any of the variables of this equation involves recomputation for all feet in the zone, which can be in the thousands. Also, the system needs to go beyond recomputation; it must reinterpret and revise the status of its conclusions and recommendations.

An expert analyst usually has heuristics on the use of this equation. These heuristics suggest that this formula is appropriate for a given situation, or that other techniques should also be performed if this method is used. Heuristics also suggest what parameters should be monitored, so that when they change this equation is re-invoked, and what interpretations should be made when this equation is used. It is these types of heuristics that are captured in the production rule model and provide ELAS with its interpretive capabilities.

Based on the structure of ELAS, a more generalized representation can be presented for building an expert system in other applications, Making interpretations of observations and tests is a common activity of expert systems. However, when these observations are constantly changing, the interpretation strategy needs to be far more dynamic. This is the scenario in ELAS, where the integrated package is automatically passing back arguments from the methods when they are invoked. When an argument is passed, the interpretations are updated.

Amoco's log-analysis software requires a potential user to have sufficient knowledge of both the use of this software and the techniques of problem solving in well-log analysis. ELAS helps provide this expertise. The integration of the production rules and mathematical methods allows for explicit representation of rules that monitor the methods. The information acquired through this monitoring is used to provide dynamic guidance to the user.

Here are some sample rules in ELAS.

```
IF:    POROSITY done and SW not done
THEN:  Advise Compute Sw to determine water saturation
         and hydrocarbons in zone.
```

The above rule represents the type of knowledge that can be classified into a set of action-recommendation rules that give advice on the appropriateness of using a method in terms of the user-supplied background data, the user-performed methods in the analysis, the outcomes of the applied methods, and incomplete steps in the analysis.

It is very common in well-log analysis to use several related methods in a specific sequence. Due to the vast amounts of data that quickly accumulate, from the user and from the results of different methods, it is difficult for a user to keep track of the correctness and consistency of an analysis sequence. ELAS keeps track of events by checking through an explicit set of production rules. The main function of these rules is to monitor and propagate dependency relations through the analysis, and examine consistency between expected parameter values and computed parameter values. For example, a user may choose to perform a certain method, without realizing that specific tasks have to be carried out to keep the analysis consistent. Production rules are used to direct these automatic updates, and a typical example is

```
IF:    Sw successfully computed
THEN:  Perform Sw goodness of fit.
```

This class of rules is categorized as sychronization rules.

While analyzing a well, a user may go through a complex process of discriminating neutron and density logs, choosing a parameter called Rw through an iterative statistical procedure, and selecting a certain porosity log through a mathematical interpretation model. If one wants to re-invoke the discrimination task on the density log, the entire chain of subsequent methods invoked is affected, and the system must make this chain of events consistent with the change. This is a small example of the many dependencies that exist in an analysis. Production rules indicate such relations and direct the propagation of associated changes. One example is as follows:

```
IF:    WATER_FIT not normal and WATER_FEET more than 20
THEN:  Indicate Bad water zone fit; try and choose alternate
         discriminators using methods A or B.
```

Production rules are used to compare the expected values of parameters and the results of applying specific methods. The user has the choice to enter certain a priori information about the problem, such as whether one ought to expect gas in the well.

```
IF:    GAS expected and method C verifies GAS and HYDROCARBON FEET equal 0
```

THEN: Indicate Hydrocarbon computation inconsistent with the amount of gas detected.

In this case, if we were told to expect gas, and gas is indicated by an analysis of some of the logs, but the overall analysis of total hydrocarbons indicates that no gas is present, we then proceed to get clues as to whether the method of analysis might be at fault, whether the logs are noisy or otherwise inaccurate, or whether some underlying assumption is unjustified, etc. These are examples of the kind of checking that is expressed through consistency rules.

All of these rules are part of a structured production system. This production rule model is invoked each time a mathematical method is performed. The control routine of ELAS, using its knowledge about the rules and their purpose, gathers all the goals concluded by the production rules, and uses them to carry out the functions of adding interpretations to the problem solution, guiding a user on what to do next, pointing out inconsistencies, and maintaining internal consistency by self-performing of dependent actions.

The components of the mathematical methods play a crucial role in ensuring that all these rules work. In a sense, the real knowledge of the domain lies in these methods, and a human expert's perspective of these mathematical methods is used to extract the necessary and sufficient set of parameters from these methods for the purpose of controlling and intepreting them. We can see that this is a necessary condition for ensuring that such a system can indeed be built; an existing software package should be able to be broken down into a set of methods that can be controlled and interpreted through precise sets of controlling and observed parameters.

The production rules may be viewed as containing consistency, control, and interpretive knowledge that is organized around methods of analysis used by an expert log analyst. This knowledge comes into play if the user invokes that method, or the state of the analysis indicates the appropriateness of the use of a method, or if the method is automatically invoked by the system due to a triggering effect.

ELAS is currently being used in a research environment for formalizing and integrating knowledge from different experts of Amoco's different regions of exploration and production. Additional efforts are underway to make available this form of analysis to Amoco's practicing well-log analysts in the field.

In order to achieve higher performance expert systems, we will likely need to use representations beyond production rules, such as mathematical and quantitative methods. These methods may already exist in highly developed software packages, and in such cases, we can take advantage of the years of developmental work. For domains where such software packages exist, we have proposed a hybrid scheme organized around mathematical methods and production rules. We have successfully implemented ELAS using a domain-specific hybrid scheme. This structure may prove to be suitable for use in building expert systems for other domains with similar problem-solving scenarios.

Although there have been attempts to build specific expert systems using available applications software, there are no general purpose system tools for these tasks. We are currently examining various approaches to generalizing the techniques needed for construction of hybrid systems. The hybrid system appraoch used for ELAS is a pragmatic first step toward the realization of this goal.

MUDMAN

An example of how AI methods are helping people solve difficult problems in the commercial sector is NL Baroid's expert system MUDMAN. NL Baroid is a $400-million-per-year company whose product is drilling mud, a lubricant needed in drilling oil wells. Baroid invented drilling mud in the 1920s and is now the largest mud company in the world.

Sometimes when Baroid sells drilling mud to an oil company, it also sells the services of a mud engineer to stay at the site and solve problems. It takes about three years to train a mud engineer. Baroid has a knowledge base of over 60 years of mud experience, both in written reports and in the knowledge of mud experts with 30 to 40 years in the field. They wanted to make that knowledge available to mud engineers in the field.

When mud engineers call upon personal knowledge to solve a problem, a plausible mechanism to describe this process is that they search through their memories, matching the current situation to a previous pattern. Then they may apply rules of thumb to solve the problem. No conventional algorithms are universally applicable, so no conventional program can reproduce the needed expertise and solve the business problem. A system like MUDMAN, however, uses AI techniques of pattern matching and the application of heuristics, or rules of thumb, to solve the problem the way an expert would, using this model of the thinking process.

The inputs to MUDMAN include the specifications of the type of mud needed in a pareticular well and the chemical and physical properties of the mud that is actually present. MUDMAN compares the specifications to the actual properties, provides an analysis of drilling problems, and recommends corrective treatments.

MUDMAN was developed in a joint effort by Baroid and Professor John McDermott (acting department head and principal scientist Department of Computer Science, Carnegie-Mellon University) and associates at Carnegie-Mellon (CMU). The developers at CMU did the feasibility studies and applied research and, during the initial joint development period, trained Baroid personnel in AI techniques. Since CMU turned over the framework for MUDMAN to Baroid in January 1984, Baroid has had full responsibility for field test, updating, enhancement, and modificaiton.

MUDMAN wsa specifically designed for sale to Baroid's customers, which are oil companies. Baroid has described MUDMAN as the first expert system sold as a commercial product to the oil industry.

SOME OBSERVATIONS ON THE TRADITIONAL WISDOM

The following material is taken from the Schlumberger references on the DIPMETER ADVISOR. They are extremely candid comments, especially when coming from the Schlumberger Research group who do not often discuss their internal failures or politics, even privately. The comments indicate both the learning curve and disappointment curve during the evolution of the project. It is reproduced in this report to emphasise the experimental and research nature of our task, and the necessity of maintaining an open mind and critical attitude toward all facets of the work.

A common maxim of expert system development is that we should throw away the code for the Mark-I version of the system as soon as it demonstrates feasibility and get started on Mark-II. In the commercial environment, there is great reluctance to throw away code. As a result, a more likely scenario involves a series of progressive releases of the system to the expert and possibly to the engineering organization for development and use.

The fact is that even though the knowledge engineer knows all too well the limitations of Mark-I, and even has ideas on how to overcome them, Mark-I may still provide some useful service. This is a good illustration of a conflict that can arise as a result of somewhat different goals of research and of development in expert systems. The former is concerned with continued exposition and machine implementation of human expert reasoning methods, while the latter is concerned with construction of products that utilize already understood and implemented methods. We do not yet know how to manage this type of progressive and evolutionary technology transfer. (The problem exists in conventional program development as well, as we have experienced with LOG/MATE ESP.)

It is well accepted that expert system development is an incremental process. Usually we understand this to mean that the performance of the system improves incrementally. There is, however, another kind of change that may occur; namely, our experts are themselves moving targets--partially as a result of the perspective gained through experience in expert system development! This has been apparent during the Dipmeter Advisor project. For example, we have seen an increasing geological awareness in our expert dipmeter interpreter. This has led to a series of changes in the way stratigraphic analysis is handled in the system. Not all of these changes have proved useful--the expert appeared to be using the program at times as a test bed for his own evolving ideas.

It is traditional wisdom that the task should be very carefully defined before the system is designed. Our experience has been that this is quite difficult. In consonance with our comments on the rapid prototyping development strategy, it is not clear that task definition can be done in a rigorous fashion. We suggest a contingent definition--one that is clear for a time, but can be easily changed. We should note that the evolving performance of the system itself at least partially fuels changes in the task definition.

It is generally accepted that construction of the Mark-I system should be commenced as soon as one example of the intended behavior is understood. We now believe that we spent too much time in knowledge acquisition before actually starting to build a system. This had the effect of slowing our rate of progress. We could not move forward in formalizing the knowledge that had been gained, because we could not demonstrate in concrete terms our understanding of it.

Some of the development team also deemed themselves to have acquired more expertise than was warranted. This is a natural tendency. It was partially due to infrequent interactions with the expert. More responsibility fell on the shoulders of the knowledge engineers to organize the domain knowledge than appears prudent. This infrequency also led to a problem of validation--how to be sure that we were on the right track. On a related note, we can testify to the necessity of an adequate set of generic examples with which to test the system as it evolves.

It is common to deal with a single expert during the development of an expert system. The perceived danger is that it is difficult enough to capture what a single expert is doing, let alone a number of experts. In the particular context of dipmeter interpretation, however, it might have been useful to involve a number of different experts from the outset. We now understand that there are many schools of thought on the problem. There is also a variety of perspectives that can be brought to bear on it--dipmeter interpretation expertise and geological expertise are not necessarily co-located in the same person.

While the rules for a first approach are most appropriately phrased by a dipmeter interpreter, we might have been well-advised to obtain the necessary geological vocabulary and structure from a geologist. In future systems, we will attempt to synthesize these overlapping points of view.

In a similar vein, we have noted a difficulty that can arise when a single expert is used and when he provides all examples with which to test the system. When working with familiar examples our expert does indeed appear to apply forward-chained empirical rules--kind of compiled inferences. Recently, however, we have participated in experiments with a number of interpreters (and examples) from around the world. During these experiments we noted that our expert resorted to a different mode of operation when faced with completely unfamiliar examples. He appeared to reason from underlying geological and geometric models--abandoning the rules.

In some sense, this is of course to be expected. It was instructive, however, to actually document the change. We believe that dealing with multiple experts would have provided concrete evidence of this phenomenon much sooner in the life of the project. Actually seeing the change in reasoning was further complicated by the fact that our expert has extremely broad experience. Hence, finding a completely unfamiliar example was quite difficult.

We have also noted a lurking danger in dealing with experts. It appears to be possible to give an expert a false sense of comfort with a particular formalism (eg., rules). At times we had a sense that the expert was trying to make us happy by expressing what he was doing in terms of the rule framework we had offered--perhaps at the cost of accuracy. We would be well-advised to avoid over-reliance on the rule (or any other presently known) framework. We don't want to convince the expert that this simple idea covers everything he does, or that system failures are necessarily the result of incorrect or missing rules.

With regard to acceptance of the expert systems approach, our experience has been somewhat different from that of the XCON (R1) designers at Digital Equipment; that is, for R1 there was general relatively rapid acceptance of the ideas within the organization. From early in the project concerns revolved almost totally around performance and utility in the problem domain.

We have seen a substantial increase in the size of the rule base (approximately tripled) and the functionality required of the system before we could consider field evaluation. This is similar to the experience with R1. The size of its rule base also tripled during the development phase.

The traditional wisdom notes the importance of early construction of a flexible user interface. For the Dipmeter Advisor system the interface is graphical. It has proved invaluable in testing and user acceptance. Furthermore, expert systems that are actually used by people trying to solve problems in their own domains of interest (as opposed to being used by researchers as vehicles for experimentation with AI techniques) must pay particular attention to human interface issues.

For the Dipmeter Advisor system, it was only after we constructed a personal workstation implementation that was flexible, robust, and fast that it became possible to seriously consider testing by the Schlumbrger engineering organization.

One final observation worth noting relates to the impact of an expert system on the domain experts. As has been found in other applications of expert systems, the existence of an expert system is helping to identify the real knowledge used in the field--the kind of knowledge that is rarely found in textbooks. A program that captures some of it at least gives a concrete basis for comparing the methods of different experts. It can also help a group to reach some form of consensus. The Dipmeter Advisor system has stimulated an examination of current dipmeter interpretation methods that promises to improve quality.

APPENDIX 1 - DEFINITIONS OF INFERENCE AND SEARCH TECHNIQUES

The following material is taken from:

> An Overview of Expert Systems
> W.B.Gevarter
> National Bureau of Standards, Washington, DC
> May, 1982 NBSIR 82-2505

A. EFFICIENT SEARCH MECHANISMS

1. Forward Chaining

When data or basic ideas are a starting point, forward chaining is a natural direction for problem solving. It has been used in expert systems for data analysis, design, diagnosis, and concept formation.

2. Backward Chaining

This approach is applicable when a goal or a hypotheses is a starting point. Expert system examples include those used for diagnosis and planning.

3. Forward and Backward Processing Combined

When the search space is relatively large, one approach is to search both from the initial state and from the goal or hypothesis state and utilize a relaxation type approach to match the solutions at an intermediate point. This approach is also useful when the search space can be divided hierarchically, so both a bottom up and top down search can be appropriately combined. Such a combined search is particularly applicable to complex problems incorporating uncertainties, such as speech understanding as exemplified in HEARSAY II.

4. Event Driven

This problem solving direction is similar to forward chaining except that the data or situation is evolving over time. In this case the next step is chosen either on the basis of new data or in response to a changed situation resulting from the last problem solving step taken. This event driven approach is appropriate for real-time operations, such as monitoring or control, and is also applicable to many planning problems.

B. Search Control and Transformation Mechanisms

Many straightforward problems in areas such as design, diagnosis, and analysis have small search spaces, either because the problem is small or the problem can be broken up into small independent subproblems. Often a single line of reasoning is sufficient and so backtracking is not required. In such cases, the direct approach of exhaustive search can be appropriate, as was used in MYCIN and R1.

1. Generate and Test

Search is often formulated as "generate and test" - reasoning by elimination. In this approach, the system generates possible solutions and a tester prunes those solutions that fail to meet apprpriate criteria. Such exhaustive reasoning by elimination can be appropriate for small search spaces, but for large search spaces more powerful techniques are needed.

2. Hierarchical Generate and Test

A hierarchical generate and test approach can be very effective if means are available for evaluating candidate solutions that are only partially specified. In these cases, early pruning of whole branches (representing entire classes of solutions associated with these partial specifications) is possible, massively reducing the search required.

Hierarchical generate and test is appropriate for many large data interpretation and diagnosis problems, for which all solutions are desired, providing a generator can be devised that can partition the solution space in ways that allow for early pruning.

3. Dependency-Directed Backtracking

In the generate and test approach, when a line of reasoning fails and must be retracted, one approach is to backtrack to the most recent choice point (chronological backtracking). However, it is often much more efficient to trace errors and inconsistencies back to the inferential steps that created them, using dependency records as is done in MOLGEN. Backtracking that is based on dependencies and determines what to invalidate is called dependency-directed (or relevant) backtracking.

4. Multiple Lines of Reasoning

This approach can be used to broaden the coverage of an incomplete search. In this case, search programs that have fallible evaluators can decrease the chances of discarding a good solution from weak evidence by carrying a limited number of solutions in parallel, until the best solutions is clarified.

5. Breaking the Problem Down Into Subproblems

This approach (yielding smaller search spaces) is applicable for problems in which a number of non-interacting tasks have to be done to achieve a goal. Unfortunately, few real world problems of any magnitude fall into this class.

For most complex problems that can be broken up into subproblems, it has been found that the subproblems interact so that valid solutions cannot be found independently. However, to take advantage of the smaller search spaces associated with this approach, a number of techniques have been devised to deal with these interactions.

Sometimes it is possible to find an ordered partioning so that no interactions occur. The R1 system for configuring VAX computers successfully takes this approach.

A technique called least commitment coordinates decision-making with the availability of information and moves the focus of problem-solving activity among the available subproblems. Decisions are not made arbitrarily or prematurely, but are postponed until there is enough information. In planning problems, this is exemplified by methods that assign a partial ordering of operators in each subproblem and only complete the ordering when sufficient information on the interactions of the subproblems is developed.

Another approach, used by MOLGEN and called constraint propagation, is to represent the interaction between the subproblems as constraints. Constraints can be viewed as partial descriptions of entities, or as relationships (subgoals) that must be satisfied. Constraint propagation is a mechanism for moving information between subproblems. By introducing constraints instead of choosing particular values, a problem solver is able to pursue a least commitment style of problem solving.

6. Guessing or Plausible Reasoning

Guessing is an inherent part of heuristic search, but is particularly important in working with interacting subproblems. For instance, in the least commitment approach the solution process must come to a halt when it has insufficient informatipon for deciding between competing choices. In such cases, heuristic guessing is needed to carry the solution process along. If the guesses are wrong, then dependency-directed backtracking can be used to efficiently recover from them. EL and MOLGEN take this approach.

7. Top Down Refinement

Often, the most important aspects of a problem can be abstracted and a high level solution developed. This solution can then be iteratively refined, successively including more details. An example is to initially plan a trip using a reduced scale map to located the main highways, and then use more detailed maps to refine the plan. This technique has many applications as the top level search space is suitably small. The resulting high level solution constrains the search to a small portion of the search space at the next lower level, so that at each level the solution can readily be found. This procedure is an important technique for preventing combinatorial explosions in searching for a solution.

8. Hierarchical Resolution

Certain problems can have their solution space hierarchically resolved into contributing subspaces in which the elements of the higher level spaces are composed of elements from the lower spaces. Thus, in speech understanding, words would be composed of syllables, words, phrases of words, and sentences of phrases. The resulting heterogenous subspaces are fundamentally different from the top level solution space. However, the solution candidates at each level are useful for restricting the range of search at the adjacent levels, again acting as an important restraint on combinatorial explosion. Another example of a possible hierarchical resolution is in electrical equipment design whre subcomponents contribute to the black box level, which in turn contribute to the system level. Similarly, examples can be found in architecture, and in spacecraft and aircraft design.

9. Employing Multiple Models

Sometimes the search for a solution utilizing a single model is very difficult. The use of alternative models for either the whole or part of the problem may greatly simplify the search. The SYN program is a good example of combining the strengths of multiple models by employing equivalent forms of electrical circuits.

10. Meta Reasoning

It is possible to add additional layers of spaces to a search space to help decide what to do next. These can be thought of as strategy and tactical layers in which meta problem solvers choose among several potential methods for deciding what to do next at the problem level. The strategy, focusing and scheduling meta rules used in CRYSALIS and the use of a strategy space in MOLGEN fall into this category.

APPENDIX 2 - THE TOOLS OF THE TRADE

The following material is taken from:

An Overview of Expert Systems
W.B.Gevarter
National Bureau of Standards, Washington, DC
May, 1982 NBSIR 82-2505

Artificial Intelligence in Business
Paul Harmon and David King
John Wiley and Sons
New York, 1985

LISP

AI is considered a young field, but LISP, its premier programming language, is, relatively speaking, an old-timer. The only older programming language still in use is Fortran. While Fortran was designed primarily for numerical computation, LISP was designed primarily for manipulating symbols. Symbolic processing languages, such as LISP, extend our ability to use computers from the relatively smaller realm of numeric problems to the larger realm in which we work in words and symbols.

LISP, developed by John McCarthy (now professor of computer science at Stanford University) at the Massachusetts Institute of Technology in the late 1950s, stands for LISt Processor. LISP programs consist of collections of procedures in list form that operate together to accomplish a given purpose.

A LISP list is a string of "atoms," the basic elements that the system will manipulate, enclosed by parentheses. A list can be empty or can consist of either atoms (such as numbers, symbols, or words) or other lists.

The ease and power of recursion in LISP programs are notable. When you have solved a portion of a problem, and the problem-solving method is applicable to the remaining portion of the problem, LISP allows you to define a function to use itself repeatedly on subproblems.

LISP is a flexible language that you can modify for your own needs. You can write code ranging from operating systems to high-level programs in LISP. In fact, LISP itself can be written in LISP. LISP makes no distinction between lists that contain data and lists that contain programs. This makes it easy for LISP programs to manipulate or even generate other LISP programs. In addition, it is possible to integrate data and information about procedures.

This integration forms the basis for sophisticated "frame" and "object-based" systems commonly used in AI applications.

COMMON LISP is the de facto standard LISP and provides a base dialect from which other implementations can stem for use on personal computers, commercial timeshare computers, and supercomputers. For this reason, COMMON LISP omits features that are useful only on some classes of processors.

PROLOG

Prolog, which was named for PROgramming in LOGic, was developed at the University of Marseilles by Professor Alain Comerauer and his colleagues in the early 1970s. Other centers of Prolog development have been London and Budapest.

Like LISP, Prolog was designed for the manipulation of symbols, and both languages lend themselves to expressing predicate calculus logic. Prolog is also interactive, like LISP. Prolog, however, is characterized as a relation-processor rather than as a list-processor like LISP. Prolog was intended for use in natural language processing systems, but has also shown its usefulness in the areas of computer-aided architectural design, expert systems, and database building and query systems.

Prolog is designed in such a way as to automate search through a tree-structured domain or knowledge base. Since many knowledge bases have treelike shapes, Prolog has naturally been applied to language and query applications.

Prolog shows some promise as a suitable language for parallel processing systems now beginning to be developed. In a masively parallel processor (MPP), each node in the tree structure would be assigned to a separate processor. A Prolog application would propagate through the MPP by passing messages to activate links to nodes. In a parallel processor with only a few nodes, a Prolog program could assign tasks to processors each time the program reaches a branch point.

While LISP has been the language of choice for artificial intelligence in the United States, Prolog has been the leader in Japan, France, the United Kingdom, and Hungary. One reason may be that Prolog programs are smaller and are easier to read than equivalent LISP programs. Another reason is that Prolog's logic-based semantics hold the promise of helping to simplify the representation of knowledge. If Prolog is an inherently parallel-processing language as many contend, then it is a good fit for the parallel-processing computers the Japanese hope to build in their Fifth-Generation Computer Project.

EMYCIN

EMYCIN is the oldest system building tool in the AI environment, and it has been moderately successful.

As they finished their work, the developers of MYCIN, an expert system for diagnosing bacterial infections, realized that there wer two distinct parts to their system: the knowledge base, which was specific to the area of medical diagnosis, and the inference engine, which was a general-purpose back-chaining rule evaluator. This distinction led to building an empty MYCIN, or EMYCIN--a MYCIN without its knowledge base.

EMYCIN is a tool for building MYCIN-like consultation systems. EMYCIN expects knowledge to be represented as objects, attributes, values, and rules in much

the same way that a spreadsheet program expects its data to form rows and columns. EMYCIN contains all machinery needed to reason over a knowledge base and to conduct consultations with a user. Over the years, editors and debugging aids were added to assist a knowledge engineer in building the system. EMYCIN is a knowledge system without any domain knowledge.

EMYCIN is a tool and not a computing language. It is less general than LISP or INTERLISP, in which it was written. LISP is a general-purpose list porcessing language, whereas EMYCIN is a special-purpose O-A-V/rule processor that uses backward chaining.

OPS5

The OPS5 language was created by Dr. Charles L. Forgy, a research computer scientist at Carnegie-Mellon University, in the late 1970s for building large, forward or backward-chaining, production-based expert systems. Because of this emplasis, OPS5 is not considered a general purpose language such as LISP or Prolog. The OPS4 version of the language was written in LISP. OPS5, which Forgy wrote to be easier to read and maintain, has had three different interpreters, written in BLISS for VAX mainframes, and MACLISP and Franz LISP for smaller computers.

A production system is a program consisting of condition/action rules phrased in IF...THEN style. The knowledge base of an expert system written in OPS5, called production menory, consists entirely of production rules expressing knowledge about a problem domain. OPS5 programs have two other components: a database called working memory and the interpreter, referred to as the inference engine, which is the part of the system that selects and executes the appropriate rule at each point in processing.

At the start of the program, the user enters data and parameters relevant to the current instance of the problem to be solved into working memory. As processing moves along, working memory changes to reflect new information inferred by the system at each step.

The inference engine evaluates all of the rules to see which have IF portions that are exactly satisfied by the current state of the working memory. This set of rules is called the conflict set. If there are two or more satisfied IF's, the inference engine will act on whichever one its built-in protocol selects. This process is known as conflict resolution. Examples of conflict resolution strategies are "Fire the rule with the most precise (or complex) set of conditions" and "Fire the rule that references the newest data." Upon conflict resolution, the appropriate rule fires, that is, the THEN portion acts to change the working memory. Because this action changes the working memory, on the next round when the inference engine evaluates rules, there may be a new conflict set.

This cycle of recognizing the appropriate rule to fire, based on the updated contents of working memory, and acting by firing the rule to change working memory, continues until a conclusion is reached; that is, until the conflict set is empty or a rule halts the program. The problem solution or conclusion is represented by the final state of working memory.

The flow of control in an OPS5 program is not determined by the order in which

the programmer puts the rules in the system. Rules become candidates to fire when the "If" statement is satisfied by information in working memory.

In a conventional programming language, the order of the instructions in the program is important; if you have to add a new instruction to the system, an error in its location can change the entire function to yield a wrong result.

The sequence in which rules are written in an OPS5 program is not important. Program execution does not rely on rules being in any particular order. This makes adding new information to an OPS5 program relatively easy.

OPS5 is most often used as a forward-chaining language, which makes it appropriate for expert systems whose solutions can be reached by asking, "Given these facts, what follows?" This mode of operation has been useful in systems like XCON (previously called R1), the computer configuration system Professor John McDermott of Carnegie-Mellon University designed for Digital Equipment Corporation. McDermott also employed OPS5 in MUDMAN, an expert system, available from NL Baroid of Houston, for analyzing problems related to an oil-well drilling lubricant. A similar program, called DRILLING ADVISOR. was developed by Elf-Aquitaine using KES-300, a derivitive of EMYCIN, described earlier.

ART--THE AUTOMATED REASONING TOOL

The Automated Reasoning Tool (ART) from Inference Corporation is a tool kit for knowledge system development. The kit contains four major components: a knowledge language for expressing facts and relationships; a compiler for converting the knowledge language into LISP; an applier, which is an inference engine; and a development environment, which includes debugging aids and trace functions.

ART is a very general tool applicable to many problems. For example, ART supports time tagging within its inheritance functions. This suggests that ART can be used to build systems that reason about time-dependent events. unfortunately, there are no examples of ART in action in a commercial setting. Examples in the ART materials are about very small, "toy" systems.

ART provides a number of representations to store and maintain facts. One is the traditional O-A-V triplet. A second means of representation, called a fact, is a proposition with a truth value and a scope. Quantifiers (ie., "There exists at least one ..." and "For all...") are supported by ART. Inheritance is represented by logical linkages among objects and facts. Also, attributes and values can be inherited by parent objects in a hierarchy. Prototypical classes can be defined with default values that change only if necessary.

The inference engine or knowledge applier is described as being an opportunistic reasoner. This means that ART can reason with both forward and backward chaining, or with explicit procedural commands. Rules affect the direction of inference. In this way the inference engine moves opportunistically, depending on the pattern of intermediate results. ART also supports confidence ratings. Like all of the hybrid tools, ART is a very powerful programming environment that can, in the hands of a skilled knowledge engineer, be made to perform in a variety of different ways.

ART has a wide variety of interface features, all oriented toward helping the knowledge engineer develop an expert system. The tool is flexible enough that a skilled knowledge engineer can ues ART to develop whatever usr interface is desired.

ART is written in LISP and runs on LISP machines produced by Xerox and Symbolics. ART is available from Inference Corporation, Los Angeles, California. An initial copy costs $60,000; a second copy can be purchased for $20,000. ART can be leased for $1000 per month, or for $3000 per month with an option to buy it at the end of six months.

KEE--THE KNOWLEDGE ENGINEERING ENVIRONMENT

The Knowledge Engineerng Environment (KEE) is an integrated package of software tools available from IntelliCorp (formerly IntelliGenetics). IntelliCorp was the first knowledge engineering company, founded in 1980, and its original goal was to market genetic engineering software. In August 1983 IntelliCorp began selling KEE, a hybrid tool derived from its work with genetic engineer software. KEE, is therefore, a tool that was originally derived from complex analysis and planning applications. An enhanced version of KEE was announced in August 1984.

KEE has been used to build a number of widely used genetic engineering knowledge systems. These systems offer advice about the design of molecular genetics experiments. KEE has also been used to develop an integrated interface to a nuclear magnetic resonance spectrometer, an application that demonstrates that KEE can be embedded in laboratory equipment. KEE is popular with several R&D groups that are currently working on prototype applications. Arthur D. Little Co., for example, is working with several of its clients to develop KEE-based planning applications.

KEE's basic representational paradigm is frames, which unify the procedural and declarative expressions of knowledge. KEE is an example of object-oriented programming. Facts and rules in KEE are represented as objects or frames that have labled slots containing either values or means for obtaining values. Slots can contain a number of different entities. A slot may contain a procedural attachment, that is, a set of instructions that compute a value for a slot. Similarly, a slot may contain a set of rules that conclude values for other slots in the frame. Procedural knowledge can also be inserted in a slot as a LISP program. A slot may also point to another frame and indicated an inheritance relationship.

Redundant information entry is minimized with inheritance hierarchies. A knowledge engineer can build a knowledge base hierarchy by initially specifying generic objects and their attributes. Then, when specific objects are created, they will automatically inherit attributes of the generic objects in the knowledge base. As a result, the knowledge engineer needs to focus on only a nominal number of unique attributes for each new object.

KEE integrates frame-based and rule-based reasoning techniques to describe structures and behaviors quickly. The frame-based system enables one to include descriptive and procedural knowledge with each object. KEE allows one to define class member and subclass relatiopnships so that each link type has a uniform semantic interpretation throughtout a knowledge base.

Because user interface commands run as separate processes, the knowledge engineer can change the value of any attribute of any object while the rule system is running and can also browse through and display different objects in the knowledge base. Rules and objects can be easily identified and enhanced to improve the system's performance.

Graphics are linked to the underlying knowledge base to help explain the representation, reasoning, and behavior of a knowledge system. Using KEE's graphics editor menus, the knowledge engineer can design and construct graphic models of physical objects such as meters and gauges to monitor specific values. In the tradition of object-oriented programming, frames (as objects) communicate with one another by sending messages to one another. A message might be a request to display information or to execute a set of rules.

The inferencing scheme for KEE is quite flexible. It can be programmed to behave as a back-chainer or a forward-chainer. Values in slots can be manipulated, and results ripple throughout the logical structure of the knowledge base. Such values are called active values.

There is no sharp distinction between the consumer and creator of a KEE knowledge base. This is just another way of saying that KEE is a hybrid system that requires considerable sophistication on the part of the knowledge engineer or user. The user interface provides a number of graphics features that aid the knowledge engineer in the development and debugging of a knowledge system.

For example, when the knowledge base does not contain information on the state of an attribute, and there are no rules enabling the system to determine values from other knowledge, the system will prompt the user for an answer. The user, in turn, can ask the system why it needs to know, and see the line of reasoning leading to the questions. The line of reasoning for any set of conclusions can be shown graphically, so the user can determine how a conclusion was reached. A graphics display of the decision process adds credibility to decisions that are correct and visibility to decisions that are not.

KEE is a hybrid system and therefore can be extended by the knowledge engineer. Many of KEE's functions are defined by KEE System Knowledge Bases. Thus, the same processes used to build a knowledge system can also be used, for example, to modify existing inheritance rules or create new ones.

KEE is supported by IntelliCorp in several ways. A three-day training program is standard and further support is available after an application project is underway. In addition, on-site consulting by an IntelliCorp knowledge engineer is included with purchase of the tool.

KEE is implemented in LISP and is available on the Xerox 1100 machine, the Symbolics 3600 machine, the LMI LAMDA, and the TI Explorer. KEE can be purchased from IntelliCorp for $60,000. The cost declines rapidly for multiple copies.

LOOPS

LOOPS is a knowledge engineering environment developed at the Xerox Palo Alto Research Center (Xerox PARC). LOOPS is a software tool that incorporates a variety of different knowledge engineering constructs in one unified package. The constructs include object-oriented programming, the use of active value, a knowledge base management scheme, and a rule package.

A great number of the features of LOOPS are related to support available in the INTERLISP environment. For example, it is possible to review programs under development in LOOPS with a variety of graphical schemes. These schemes are essentially schemes provided by INTERLISP by means of its windows and break packages. After a rule is fired, it is possible to examine what the rule was, what the effect was, waht the effect of its firing was, and to trace down aspects of reasoning that take place during a program's execution.

A second construct incorporated in LOOPS is the idea of an active value. An active value operates like a probe. By examining an active value, one can see the current status of a variable being reasoned about. By attaching a graphic picture, such as a gauge or thermometer, one can see an analog representation of a variable that is being reasoned about. Moreover, one can monitor changes in that value as processing continues . By changing an active value, one can view a series of side effects associated with that value's changing.

Object-oriented programming is an orientation toward viewing the entities in a program as objects (or units or frames) that communicate with each other via messages. Attached to eack frame are constructions and declarations that define and elaborate what the frame is about. When a message arrives at a frame, attachments to that frame process the message and carry out its effects.

Prodecural and rule-oriented programming is also supported by LOOPS in a conventional way. The unique aspect of having rules and procedures as part of LOOPS comes not from the procedures as such but from how they are integrated with active values and object-oriented programming.

Perhaps LOOPS should be thought of as an environment rather than a tool. In order to build an expert system with LOOPS, one must choose from a variety of different approaches and write a fair amount of code before the system begins to home in on and help to structure the knowledge system. Its utility is primarily as a software engineer's environment, where a variety of useful subroutines have been prepared and are ready to assemble; and that is strikingly different than a tool such as S.1, where a large number of design decisions have already been made and are "hard-wired" into the system.

Xerox PARC has offered a three-and-a-half day course to teach parrticipants to use LOOPS. The course is built around a game called "Truckin." Each participant develops a program to manage a "truck." Then the various truck programs compete to buy and sell commodities and avoid the hazards of the road in a simulated environment.

LOOPS is implemented in INTERLISP and runs on Xerox 1100 LISP-based personal workstations. LOOPS is available fromm Xerox for $300 !!! in an unsupported version. Xerox apparently regards LOOPS simply as a research tool and a powerful demonstration of what the 1100 series of LISP work-stations can do.

EXPERT

EXPERT is a tool for designing and building consultation systems. It paradigm is the diagnosis/prescription model (which its developer call the classification model). EXPERT was built by Sholom Weiss and Casimir Kulikowski of Rutgers University. These two individuals have used the system to develop several large and small knowledge systems.

The three examples that we outline below are described in detail in a book written by EXPERT's creators, A Pracical Guide to Designing Expert Systems (1984):

- Serum protein diagnosis program. This knowledge system examines profiles of data from a spectrum analyzer. It classifies the profiles and selects an appropriate diagnosis to display. The system was build with EXPERT and then recoded into assembly languagee to be stored in a read only memory (ROM). This ROM is installed in the spectrum analyzer. The instrument plots the profile and prints an interpretation.

- Another, Larger system built with EXPERT is a rheumatic disease consultant.

- EXPERT has been used to develop a log analysis system for oil drilling. The system is called ELAS and was developed for Amoco as a front end to their interactive log analysis package. Details are described elsewhere in this report.

EXPERT stores facts as attribute-value pairs. Facts are classified in two ways: as findings or hypotheses. Findings are observational data coming into the system. Hypotheses are potential solutions, one or more of which will be selected by the system. Relationships and heuristics are stored as production rules grouped in three categories:

- "F-F" rules that link a finding with other findings,
- "F-H" rules that relate findings to hypotheses, and
- "H-H" rules that link one hypothesis with other hypotheses.

In the ordinary course of a consultation, incoming data are interpreted and refined with F-F rules. The consolidated description of the findings are then related to the set of possible solutions with F-H rules. Finally, hypotheses are refined using the H-H rules.

The inference engine for EXPERT is questionaire-driven. Findings are sought one after another before the system begins to reason. After all information is obtained, rules are fired in the following order: F-F, F-H, and then H-H. Thus, EXPERT is not a back-chainer. Control is established by the order and category of the rules.

The system is able to reason with incomplete, and incomplete and uncertain data. A belief measure ranging from -1 to 1 is associated with facts.
The user interface is very simple. Questions are asked, answers are obtained, the reasoning process takes place, and an answer is given.

From the knowledge engineer's perspective, EXPERT is a batch-processing knowledge system tool. There is no interactive editor, nor are there other software aids for building knowledge bases. Trace facilities are available at runtime to assist in debugging.

Statistical functions are available to examine how much a rule improves performance. Modifying a rule changes the system's overall performance on a set of cases. Errors occur in two ways: false-positive diagnoses (cases that EXPERT judges faulty but are not) and false-negative ones (cases where EXPERT fails to locate fault that is present). EXPERT's statistical functions assist the system designer in tuning the performance of the system.

EXPERT is written in FORTRAN and is available for a number of different operating systems and many types of hardware. Interfaces to data bases and sensors are supported by FORTRAN. Acces to EXPERT is via Rutgers University and is subject to negotiation.

TIMM-THE INTELLIGENT MACHINE MODEL

The Intelligent Machine Model (TIMM) is a knowledge system building tool designed to be used by subject matter experts. TIMM focuses on cases that represent good examples according to the expert. In effect, TIMM works like a more complex version of Expert-Ease. Each set of examples forms a matrix and a matrix is, in effect, a rule. Each rule is created as the expert enters examples. Unlike Expert-Ease, however, TIMM enables several rules to be created and linked together. TIMM's example-oriented knowledge acquiaition system makes it especially appropriate for systems builders who want to model their own expertise. Several commercial systems have been built using TIMM.

Facts in TIMM are represented as attribute-value pairs. Rules are not entered directly but, rather, are build from stylized examples. That is, examples are entered as a set of conditions (attributes) related to a particular outcome or recommendation. TIMM is able to consolidate and generalize rules based on a set of cases. TIMM is capable of storing about 500 rules. It is able to handle problems that require a system to choose among 25 distinct recommendations based on some 50 factors, each of which can have up to 25 different values.

TIMM does not support associations and inheritance relationships, nor are there explicit ways of controlling the flow of a consultation. Inference and control decisions are made by an algorithm that optimizes a path through a decision tree that TIMM creates from the examples and rules it is given.

Inexact and incomplete information is handled in two ways. First, there is a certainty factor between 0 and 100 associated with facts. Second, there is a reliability number, also ranging from 0 to 100, that is associated with each set of examples and their conclusions. Thus, TIMM can report a soluton with certainty 50 and reliability 80.

In the case of TIMM, the knowledge system designer is a subject matter expert, not a knowledge engineer. Here is how a knowledge system building session proceeds:

- TIMM interogates the expert about what attributes matter with respect to a particular domain. Ranges of acceptable values are stored.

- TIMM requests that outcome or result for an example in the problem domain. It then probes to see what values are associated with the attributes for that case.

- TIMM generalizes rules and optimizes a decision tree based on the cases generated by the expert.

TIMM provides some debugging aids for the system builder. One is an explanation facility, which kdentifies all rules used in a consultation.

A two-day training course and maintenance is included in the price. Support services, 25 user's manuals, and on-site installation is also provided with each purchase. TIMM is written in FORTRAN. It is available for many mainframe and minicomputers such as IBM, DEC, Prime, and others. General Research has a personal conputer-based version of TIMM.

TIMM is available from General Research Corporation. Version 2.0, capable of linking together separate knowledge bases, is priced at $39,500. The IBM PC XT version of TIMM costs $9,500. TIMM is also offered via time sharing for a monthly charge of $500 plus computer time.

APPENDIX 3 - BIBLIOGRAPHY

1. The Artificial Intelligence Experience
   Susan J. Scown
   Digital Equipment Inc., 1985

2. Artificial Intelligence in Business
   Paul Harmon and David King
   John Wiley and Sons
   New York, 1985

3. Understanding Artificial Intelligence
   H.C.Mishkoff
   Texas Instruments Learning Center, 1985

4. Handbook of Artificial Intelligence (3 volumes)
   Avron Barr, Edward Feigenbaum et al
   W. Kaufman Publishing, Los Altos, 1981

5. Artificial Intelligence
   David L. Waltz
   Scientific American, Oct. 1982

6. Programming in Prolog
   W.F.Clocksin, C.S.Mellish
   Springer Verlag, 1982

7. LISP
   P.H.Winston
   Berthod-Klaus-Paul-Horn, 1983

8. Principles of Artificial Intelligence
N.J.Nillson
Tioga Press. 1982

9. Programming in OPS5
Lee Brownstone et al
Addison Wesley, 1985

10. Building Expert Systems for Controlling Complex Programs
Sholom Weiss et al
Rutgers University/Amoco, 1983

11. An Approach to Expert Control of Interactive Software Systems
Sholom Weiss et al
Proc IEEE, 1985

12. The Dipmeter Advisor: Interpretation of Geologic Signals
Randall Davis et al
Proc IJCAI, 1981

13. The Dipmeter Advisor System
Reid Smith and James Baker
Proc IJCAI, 1983

14. Applying Artificial Intelligence to the Interpretation of Well Logs
P.L.Baker and S.W.Smoliar
Proc IEEE, 1984

15. Problem Solving in The Domain of Quantitative Well Log Analysis
Stephan W. Smoliar
Proc IEEE, 1985

16. Expert Systems and Deep Knowledge
Stephan W. Smoliar
10th World Computer Conference,1985

17. An Object Oriented Approach to Planning
Stephan W. Smoliar
Schlumberger, 1985

18. Strobe: Support for Structured Object Knowledge Representation
Reid G. Smith
Proc IJCAI, 1983

19. Impulse: A Display Oriented Editor for Strobe
Eric S. Schoen, Reid G. Smith
Proc Nat Conf AI, 1983

20. A Modular Tool Kit for Knowledge Management
Gilles M. Lafue, Reid G. Smith
Proc IJCAI, 1984

21. Declarative Task Descriptions as a User Interface Structuring Mechanism
R.G. Smith, G.M.E. Lafue, S.C. Vestal
Computer Magazine, Sept, 1984

22. Implementation of an Integrity Manager With A Knowledge Representation System
R.G. Smith, G.M.E. Lafue
Expert Database Systems, 1985

23. Representation and Use of Explicit Justifications for Knowledge Base Refinement
R.G. Smith, H.A. Winston
IJCAI, 1985

24. The Design of the Dipmeter Advisor System
R.G. Smith, R.L. Young
Proc ACM, 1984

25. On the Design of Commercial Expert Systems
Reid G. Smith
AI Magazine, Fall 1984

26. The Use of Qualitative and Qualitative Simulations
Reid G. Simmons
AI Lab Report, MIT, 1984

27. Oil well Data Interpretation Using Expert System and Pattern Recognition Techniques
Alain Bonnet, Claude Dehan
IJCAI, 1984

28. Model Expert System (for log analysis)
Guan Jiwen, Xu Yung, Chang Minche, Zhao Jizhi
IJCAI, 1984

29. Development of the Prospector Consultation System for Mineral Exploration
R.O. Duda, et al
SRI Report, 1978

30. Model Design in the Prospector Consultant System for Mineral Exploration
R.Duda, J.Gasnig, P.Hart
Expert Systems and AI Applications, 1979

TABLE 1 EXPERT SYSTEM TOOLS

System Name	Available On	Language	Supplier
Small Systems			
AL/X	Apple II	Pascal	U. of Edinburgh Edinburgh, Scotland
ESP Advisor	IBM PC	Prolog	Expert Systems King of Prussia, PA, 19406

Expert/Ease	IBM PC	Pascal	Expert Software
	DEC Rainbow		San Francisco, CA, 94114
EXSYS	IBM PC	C	EXSYS Inc.
			Albuquerque, NM, 7194
Insight	IBM PC	Pascal	Level 5 Research
	DEC Rainbow		Melbourne Beach, FL, 32951
M.1	IBM PC	Prolog	Teknowledge
			Palo Alto, CA,94301
OPS5+	IBM PC	C	Artelligence
			Dallas, TX, 75240
Personal Consultant	IBM PC	C	Texas Instruments
	TI PC		Dallas, TX, 75380
Series-PC	IBM PC	Lisp	SRI International
			Menlo Park, CA, 94025
Medium Systems			
Expert	IBM	Fortran	Rutgers University
			New Brunswick, NJ, 08903
KES	IBM PC	Lisp	Software A&E
	DEC VAX		Arlington, VA, 22209
	Apollo, Xerox		
	Symbolics		