International Journal of Thermophysics, Vol. 9, No. 2, 1988

Issues in Expert System Development[1]

C. L. Baer[2]

The explicit representation of domain knowledge and its separation from the processes which manipulate it and the representation formalism particular to artificial intelligence allow expert systems to solve problems which are characterized by a high combinatoric complexity or which are sufficiently ill defined as to not have reasonable software engineering solutions. The expert system approach to problem-solving differs radically from its conventional system development counterpart. This paper defines the expert system and introduces the production system architecture. The relative strengths and weakenesses of expert system and software engineering approaches to problem solving are discussed. Also addressed are criteria for identifying problems amenable to expert system solution and some justifications for system development.

KEY WORDS: artificial intelligence; computer programming; expert system; human information processing.

1. INTRODUCTION

The expert system is the product of a relatively young technology which has only recently been taken from the research environment to be used in successful commercial and engineering applications. This paper introduces the expert system and the types of problems emanable to such a solution. This paper is introductory in nature; the interested reader can find more in-depth treatments of the topic in many of the works listed in the Reference [1–6].

2. EXPERT SYSTEM CHARACTERISTICS

The expert system can be informally defined by some generally agreed upon characteristics [3, 4, 6].

[1] Paper presented at the Ninth International Thermal Expansion Symposium, December 8–10, 1986, Pittsburgh, Pennsylvania, U.S.A.

[2] Pittsburgh Software Company, 3400 Forbes Avenue, Pittsburgh, Pennsylvania 15213, U.S.A.

0195-928X/88/0300-0283$06.00/0

2.1. Expertise

Quite obviously, an expert system must have expertise in some domain of interest. Expertise in this context means that the system demonstrates a level of performance comparable to that of a human expert. This level of performance is qualified by skillfulness, the system's depth of knowledge, and by robustness, its breadth of knowledge.

Knowledge is the key to the expert system. The skillful system, like the expert, uses knowledge to perform efficiently and effectively. This knowledge is not general problem-solving knowledge but, rather, the domain-specific knowledge gained through education and experience that distinguishes the expert from the novice. This knowledge enables the expert system to zero in on potential pitfalls in the solution process early, thereby solving problems with as little unfruitful activity as possible. For example, an expert who knows that the solution to a particular kind of problem is always most tightly constrained by one or two aspects of the problem will focus on those aspects first, and so reduce the amount of reworking which must be done to accommodate subproblem interactions which invariably occur as the solution proceeds. The same problem-solving behavior is not observable in the novice.

A problem which has many subproblem interactions is that of physically configurring computer systems. Problem aspects are things such as (1) how much memory is required, (2) how many users there will be, (3) how much secondary storage is needed, (4) what power supplies are required or available, (5) the dimensions of the room where the computer is to be kept, etc. Fulfilling each of the requirements listed above constitutes a subproblem solution. A novice might (and many salesmen did) attempt to configure systems by solving each aspect of the configuration independently. This approach is virtually guaranteed to fail since the final step of combining subproblem solutions would show that the components they proposed would be incompatable. More to the point, subproblems are generally not independently solvable, and they interact to constrain available options. Some constraints are more absolute than others. The expert is able to identify these absolute requirements and bring them to bear early in the solution process. In so doing, he reduces the number of options which must be considered and eliminates a potential stumbling block from later stages of his problem solving. Perhaps the most absolute and most constraining subproblem from the list above is the physical space available to keep the computer. This, then, is a requirement that would probably be used by the expert as an early constraining factor.

Explicit knowledge representation and the system's flexibility in manipulating it essentially distinguish the expert system from procedural

computational systems. Generally, rules of thumb called heuristics are used to rule out classes of solutions and to identify rapidly a nearly optimal solution. These heuristics are chunks of experiential knowledge particular to the expert, which can be abstracted and symbolically represented. A heuristic from the configuration problem above might be, "If the physical space available is not big enough for any system, indicate that a system cannot be configured."

It should be noted that heuristic solutions will be satisfactory but will not necessarily be optimal. Satisficing is inherent in heuristic problem solving, since by using heuristics one sacrifices the thoroughness of algorithms which guarantee optimal results.

2.2. Knowledge Representation

Knowledge representation is frequently in the form of situation-action rules called productions. Systems which use this type of rule are called production systems. They are the most commonly implemented type of expert system. The rule consists of an identifier used by the inference mechanism, a set of conditions, or antecedents, which constitute the left-hand side of the rule, and a set of actions to be performed when the rule has been chosen to be executed. These actions constitute the rule's consequent, or right-hand side. An English translation of a rule might read, "Rule-001 (LHS): If available dimensions are at least 8 ft by 10 ft, then (RHS) assert that usable box categories are A, B, and C."

2.3. The Inference Engine and the Knowledge Base

The driving mechanism for all production systems is the inference engine. As its name implies, its purpose is to drive the inference (reasoning) process. Although there are as many types of inference engines as there are types of inference, they have in common a central driving process called the recognize–act cycle. This cycle matches rule conditions to working memory, a type of global data base in which all pertinent objects, here called working memory elements, are represented. The inference engine selects one rule from the set of matched rules according to a conflict resolution strategy and executes that rule. A conflict resolution strategy is needed because more than one rule may be matched at any given time. The rules which are matched constitute the conflict set. The conflict set is so named because all rules in it are vying to be executed. A typical conflict resolution strategy is to chose the rule which has been matched by the most recently created object in working memory. The reasoning behind this is as follows: working memory elements are created as a consequence of the

execution of some rule. The existence of a new working memory element (some new information) means that a rule was recently fired. If that element in turn matches another rule, then that element has served as the first link in the newest chain of reasoning. (All solutions are chains of rules in which the final rule's action is to conclude or assert something.) Selecting rules from the conflict set by recency is the program's way of exploring the latest line of evidence. As the previous paragraph implies, the execution of a rule has the potential to change the states of working memory elements. As working memory elements change, new rule conditions are matched, and the conflict resolution strategy is again used to select the dominant rule for execution from a group of newly instantiated rules.

Any rule has the potential to be executed at any time as long as the conditions of its antecedent have been satisfied. This separation of knowledge from the driver is the real power of the expert system. Programs with this separation are known as declarative programs because the knowledge upon which they are based is explicitly declared. Where, in a procedural program, as program procedures are explicitly declared, one must account for and provide strongly sequenced courses of action for every possibility, in the declarative program, one need only provide for the existence of some condition, on concurrent conditions, at any point in time. In this sense, expert system behavior spontaneously adapts to the problems put to it.

2.4. Pattern Recognition

A consequence of declarative programming is that the same "intuitive" leaps demonstrated by the expert can be abstracted and represented at an appropriate level of detail. If it is not necessary to know the processes which underlie the expert's recognition of, and reaction to, a problem, then those processes need not be represented.

This is important because studies have shown that superior ability for pattern recognition plays a large role in expert performance [8]. Such pattern recognition is apparently not always a logical process. The only thing necessary to the system, however, is that the pattern and the expert's reaction to it be explicitly represented.

2.5. Robustness

In addition to having a deep knowledge of the problem area, the expert system should be robust, i.e., should have enough breadth of knowledge for reasonable behavior. A robust system's performance will

degrade gracefully as it is pushed to its domain limits. It will know what it does not know and will fail reasonably when given incorrect data or incomplete rules. For example, an expert system in cardiovascular disease should have a mechanism to let it quickly recognize that it cannot diagnose a problem in the oncology domain. Robustness comparable to that of the human problem solver can be achieved only by using general knowledge and problem-solving methods to reason from first principles. The body of knowledge necessary to achieve this is so large that very robust systems are not yet available. This is due in part to limitations of the current state of technology and in part to the relatively short time that inference processes have been under study. More robust systems should emerge as research continues and as more memory and faster processors become available.

2.6. Symbolic Reasoning

Reasoning for all intelligent systems, both electronic and biological, is a process of symbol manipulation [9, 10]. The intelligent system consists of symbols which designate things in the world, relationships between symbols, and symbol manipulation processes. In the expert system, symbols are alphanumeric strings. Their interrelationships are represented by symbol structures, and their manipulation is through various functions which are controlled by the inference engine.

2.7. Depth

A second characteristic of the expert system is that it solves difficult, challenging problems. They are developed to work in real-world problem domains, rather than in "toy" domains. Toy domains are carryovers from early work in artificial intelligence which stressed general problem solving. Because of the enormity of the tasks attempted, the domains of these early systems were necessarily trivial. Because the emphasis of expert systems is not on general problem solving but, rather, on problem solutions for narrow, knowledge-intensive domains, they are comparatively easy to build and can produce cost-effective solutions to real-world problems.

2.8. Self-Knowledge

The expert system may also be characterized by self-knowledge. It reasons about its own operation through special rules which it uses in conjunction with its own structure to help determine its actions. The knowledge a system has about how it reasons is called metaknowledge. Metaknowledge makes possible the explanation facility, one of the most innovative qualities of the expert system. This facility is part of the user

interface of many expert systems. Through it, the system can answer questions about how and why it arrived at conclusions and recommendations. It can also explain to the user why it needs the data it requests.

The benefits of an expanation facility are several. First, it ensures an improved user confidence. Many professionals are unwilling to take the advise of a machine, particularly in critical situations. For this reason, explanation facilities were built into early diagnostic systems and have remained an integral part of many systems today. The facility helps the end-user to accept the machine as a reasoning entity [11]. To enhance further man–machine rapport, explanations are generally put through a natural language translation mechanism.

Another advantage the explanation facility provides is for faster system development. The program tells its developer what it is doing, thereby significantly reducing the debugging task. The facility also facilitates predicting the effect of changes on system operation by making explicit the assumptions which underlie system operation.

3. PROCEDURAL VS DECLARATIVE PROGRAMMING

In procedural approaches, knowledge is embedded in a strongly sequenced set of instructions. One builds a program by specifying the operations to be performed in solving a problem, leaving implicit the assumptions upon which the program is based. In the production system, however, knowledge is explicit and accessible in the form of production rules. One constructs a program by describing its application area through a set of productions. The assumptions are explicit, but the choice of operations is implicit. The expert system's flexibility in problem solving stems directly from this explicit representation of domain knowledge and its separation from the driving mechanism. Procedural systems cannot fall back on a knowledge base, since their assumptions about what is true in the domain are implicit and inseparable from the instructions in which they are embedded.

Because each production contains a chunk of knowledge, the system's performance improves and its scope increases incrementally with the addition of new rules to the knowledge base. Procedural systems require generally lengthy code rewrites.

4. THE APPROPRIATENESS OF AND JUSTIFICATION FOR DEVELOPMENT

The key factors in determining the appropriateness of expert system development are the nature, complexity, and scope of the problem at hand.

The task to be performed must require manipulation of symbols and symbol structures. If symbolic reasoning is not required, then there is no need for an expert system. Chances are that the cheaper and more efficient solution would be a procedural program.

The problem should be neither too simple nor too complex. A general rule of thumb is that it should take the human expert from 15 min to 2 h to solve an appropriate problem. A problem which takes less time to solve is probably too simple to merit the effort of the knowledge engineering process, and anything which takes longer may be too complex for current state-of-the-art technology to handle.

Problem scope should be sufficiently narrow that the body of knowledge required can be explicitly represented and manipulated by the system. At the same time, its scope should be broad enough that its users find it a worthwhile tool. Failure to scope the problem correctly is the single greatest cause of system failure and underuse.

Expert system development is justifiable only if the task solution has a high payoff, human expertise is scarce or is being lost, or expertise is needed in a physically hostile environment.

5. THE COSTS OF DEVELOPING AN EXPERT SYSTEM

The resource commitments for expert system development are large. For this reason, one should have definitely ruled out the possibility of a software engineering solution before committing to expert system development. Should expert system development be in order, the following is a probable scenario. In all likelihood, a knowledge engineeer, someone experienced in building expert systems, will have to be hired to assess the feasability of expert system development. In terms of manpower, the average project will require from 5 to 6 man years. In the first year, 75% of the expert's time will be required by the knowledge engineer. It is therefore imperative that the expert have a strong vested interest in system development. The knowledge engineering process is intensive and time-consuming. Anything less than total cooperation from the expert will jepordize the project's success. A team of AI programmers will have to be hired or trained for the project. Hardware and software costs will add to the total. And finally, upon or sometime before delivery of a completed system, a training period for end-users must be provided for in the cost calculations. Ideally, to assure end-user acceptance, they should be involved in the development process as early as possible.

6. TOOLS

There are many hardware and software tools available today for expert system development, and more are introduced on the market each month. Machines which can be used for development are Lisp machines, minicomputers, and microcomputers. Of these, the most powerful are the Lisp machines. Because of their specialized architecture, Lisp machines provide the richest and most powerful Lisp processing environment. They are, however, very expensive and do not perform number-crunching tasks very well, since their hardware is set up to facilitate the processing of LISP symbol structures. Advantages they offer are built-in object-oriented graphics, a large amount of dedicated computing power, and excellent debuggers and editors which facilitate exploratory programming. Their primary disadvantages are cost and slow procedural processing.

The minicomputer with an appropriate operating system provides a good mix of symbolic and general-purpose computing. Several environments, shells, and languages for use on the mini are on the market today. These machines are particularly good when a mix of number crunching, data retrieval, and symbolic processing is required.

A final option is the microcomputer. Increased memory capacity and processing speeds have allowed it to enter the AI market. While their use must be restricted to smaller applications of approximately 50–150 rules, they do have a place on the expert systems scene. OPS-83, Smalltalk, several versions of LISP and PROLOG, and other shells and languages are today available for use on microcomputers.

Software tools fall into three categories: programming languages, shells, and development environments.

Programming languages are symbol manipulation languages. Into this category fall the many varieties of LISP and PROLOG, and OPS-83, to name a few. They offer a maximum of programming flexibility but do not provide good debugging facilities or built-in functions which might reduce program development time.

Shells are the inference engines of previously developed expert systems. Theoretically, one need only supply the knowledge base and a few functions for the desired domain. These systems can significantly reduce development time, since the inference engine, explanation facility, and user interface are already developed. However, they are useful only for applications whose characteristics are very close to those of the original expert system.

The state of the art in software tools is the development environment. These environments boast sophisicated, interactive graphics packages, debugging editors, multiple windowing, predefined symbol structures, and

other facilities to reduce development time. They are correspondingly expensive but do offer the best mix of expressive flexibility and programming support.

7. CONCLUSION

Expert system technology is just leaving its infancy; the field still holds many questions about inference and representation which research must resolve. However, the success of this technology and its impact in business, industry, and science is undeniable. They have provided cost-effective solutions to previously unsolvable problems. By consistently solving organizations' more mundane problems, they have freed their experts to tackle more interesting, challenging problems: a situation both good for the organization and good for its experts. Finally, the very process of creating expert systems has given their developers another very tangible benefit. This is the explicit representation and colation of a previously undefined body of knowledge. The unofficial credo of the knowledge engineer very succinctly describes both the essence and the promise of the expert system: Knowledge is power.

REFERENCES

1. E. Rich, *Artificial Intelligence* (McGraw–Hill, New York, 1983).
2. E. Kant, L. Brownston, R. Farrell, and N. Martin, *Programming Expert Systems in OPS-5* (Addison-Wesley, New York, 1984), pp. 5–28.
3. F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (eds.), *Buildin Expert Systems* (Addison-Wesley, New York, 1983).
4. D. A. Waterman, *A Guide to Expert Systems* (Addison-Wesley, New York, 1985).
5. D. G. Bobrow, S. Mittal, and M. J. Stefik, *CACM* **29**:880 (1986).
6. F. Hayes-Roth, *CACM* **28**:921 (1985).
7. C. L. Baer, *Technical Report—TR0986-1* (Pittsburgh Software Co., Pittsburgh, 1986).
8. W. Chase (ed.), *Visual Information Processing* (Academic Press, New York, 1973), pp. 215–281.
9. H. A. Simon, *The Sciences of the Artificial* (M.I.T. Press, Cambridge, Mass., 1969).
10. D. A. Norman (ed.), *Perspectives in Cognitive Science* (Lawrence Erlbaum Associates, Hillsdale, N.J., 1981), pp. 13–26.
11. W. G. Lehnert and M. A. Ringle (eds.), *Strategies for Natural Language Processing* (Lawrence Erlbaum Associates, Hillsdale, N.J., 1982), pp. 245–274.

Printed in Belgium