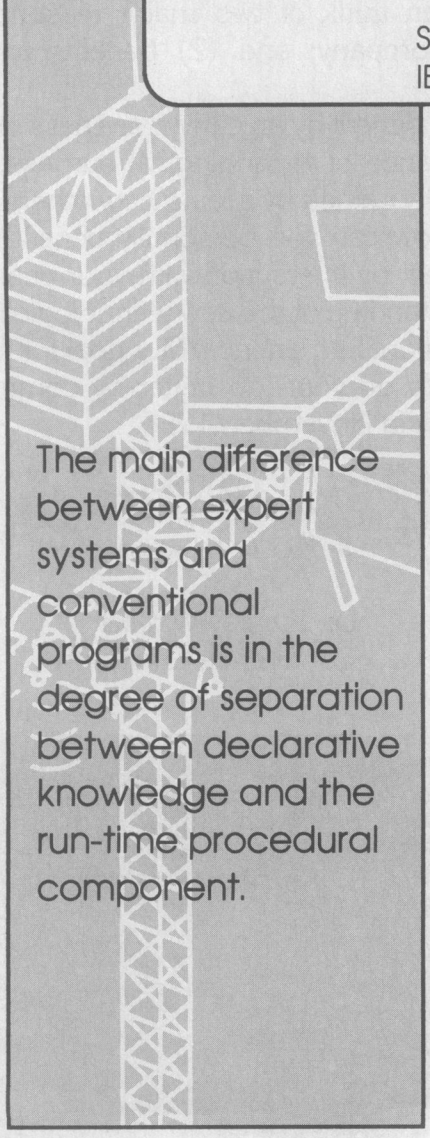
Guest Editor's Introduction

Se June Hong
IBM T. J. Watson Research Center



The main difference between expert systems and conventional programs is in the degree of separation between declarative knowledge and the run-time procedural component.

Expert (or *knowledge-based*) systems represent a relatively new programming approach and methodology, one that evolved and is still evolving as an important subarea of artificial intelligence research.

The main difference between expert systems and conventional programs lies not in the delivery of expertise, for many conventional programs can perform "expert" tasks. Rather, it lies in the way the programmer makes use of the different degree of separation between the dynamics of when to do what to perform a task on the one hand, and the "static" (that is, nonprocedural) application-domain knowledge on the other hand.

In early programs, data and code were often intermixed. Modern-day programs generally have clearer separation between data and program constructs. The expert-systems approach permits even greater separation by providing nonprocedural constructs on a still higher level. Such separation usually allows more flexibility in generating and maintaining the program, as well as greater ease of understanding, at the cost of run-time overhead necessitated by the underlying mechanisms that support it. This kind of tradeoff is well demonstrated by comparing logic simulation programs: compiled simulation versus table-driven simulation.

For many applications, especially those we usually think of as belonging to some domain of expertise, it is usually impossible to define the total flow of actions taken by a human problem-solver. Expertise is, however, often describable in bits and pieces, each relating some small situation context and the actions appropriate for the situation. Expert systems provide a mechanism (often referred to as an *inference engine*) to thread these pieces together dynamically at run-time so as to successfully complete a task. Conventional programming basically allows two simple options:

0018-9162/86/0700-0012$01.00

- test and branch, or
- do the next action in the list.

While these two are sufficient for any programming task, programmers suffer if they must write programs at this level when the domain involves expertise that is not thus resolvable.

Characteristics of an expert

An expert not only possesses a store of definitional (or *declarative*) knowledge of a domain, but also can quickly apply the knowledge to a given task. Therefore, to emulate the problem-solving skills of an expert, it is not sufficient to encode the declarative knowledge separate from the procedural component of the expertise, leaving the latter to proceed in the fixed way provided by an inference engine. A good example of this is a chess program written in a few pages of Prolog, complete with a goal to checkmate. With only the depth-first backtracking mechanism of Prolog, it would take eons to generate the moves for a winning play—hardly an expert player's behavior. Procedural knowledge in the expert-systems context addresses issues at a much higher level (that of determining the relevancy of actions and subgoals) than the notion of instruction flow found in conventional programs.

Three things are essential to being an expert. First, an expert has knowledge of the concepts relevant to the domain, its taxonomy, and interrelationships among them; if necessary, the expert can solve novel problems by reasoning from the domain principles. Second, an expert has a wealth of situation-specific, ready-made answers or partial answers that shortcut the problem-solving process. Third, an expert has the ability to recognize opportunities for using such shortcuts, and invokes the right "situation-action pair" so that focused progress is made toward completing the given task.

The state of the art

Declarative knowledge. There is a reasonably useful array of schemes to represent this first kind of knowledge, although most of them can describe only simple relationships among concepts. Theorem-proving techniques can be used for reasoning from the domain principles, if no surface heuristics are available. However, theorem-proving is inherently expensive computationally, and the burden is on the programmers to find as many shortcut rules as possible to minimize "deep reasoning." Most of the current generation of expert systems relies solely on the situation-action paradigm and gives up in those cases where only deep reasoning can provide the solution. Providing special-purpose reasoners, and representations for efficient deep-reasoning capabilities in specific domains, is an active area of AI research.

Ready-made knowledge. Handling the second type of knowledge is the forte of state-of-the-art expert systems. The situation-action knowledge is implemented as demons, as actions to be taken on an if-needed basis within a frame, and, most commonly, as explicit rules. The initial success of expert systems was largely due to their ability to capture this kind of knowledge and make inferences with it. Still, there is much progress to be made in improving the expressivity of the rule language. Inference engines that invoke these rules in sequence are currently rather inflexible, regardless of whether the sequencing is basically goal driven or data driven. Rules usually describe appropriate actions in a given situation, but whether doing the consequent action now is relevant to the task at hand is often not known. Most of the current-generation inference engines, both goal driven and data driven, provide depth-first invocation of rules, with some limited variations.

Meta-knowledge. The third kind of knowledge is perhaps the hardest to elicit from an expert—even when the programmer is the expert. Imagine rules as segments of a flowchart that is cut up into pairs consisting of a "test" diamond followed by a "procedure" box. Without knowing which segment should follow another segment, it would take an enormous amount of trial and error for someone to perform the program tasks manually—if they could be performed at all. Exaggerated though the analogy may be, essentially this difficulty is reflected in the inference mechanisms that are in use today. Meta-rules (the use of tags or flags attached to data) and explicit control languages are attempts to harness this third kind of knowledge; that is, the knowledge of when to invoke what at various levels of detail. Much of this procedural knowledge (or *meta-knowledge*), which produces focused progress in task performance, may be common to all problem-solving activities, while some must be domain specific. Either way, the means of capturing it represents perhaps the weakest part of expert systems today. Consequently, this is an active AI research area. The problem is accentuated in engineering applications that have many alternative methods and high demand for procedural efficiency.

An expert not only possesses a store of definitional knowledge of a domain, but also can quickly apply the knowledge to a given task.

The field today

Expert systems expand the use of computers to many applications for which they weren't being used at all or were being used unsuccessfully within conventional programming practices. Many successful expert-systems applications are beginning to appear in medicine, business, finance, and engineering. Also, many tools for building expert systems, in the form of shells and programming environments, are beginning to be commercially available.[1]

There have been high expectations and a corresponding flurry of activities in this area. This brought on some myths and inflated expectations about the state of the art and the limits of expert systems. Ready availability of expertise, the need for intelligent programmers, user-friendly interfaces, the sound practice of software engineering principles, and convenient development environments are no less important to expert-systems applications than to traditional programs. Automatic acquisition and compilation of knowledge

Automatic acquisition and compilation of knowledge is still a goal of AI for which research is just beginning to yield some very limited results.

is still a goal of AI for which research is just beginning to yield some very limited results. Performance and verification concerns are perhaps more serious for expert systems than for conventional programs. Yet, it is clear that more applications are, indeed, enabled by this addition to software technology. The articles in this issue represent some of these engineering applications and some efforts at advancing the state of the art so that more engineering applications will be made possible.

In this issue

The purpose of this special issue is not to give a survey or a tutorial on expert systems, but rather to share examples of engineering applications that depict the advantages, the current limitations, and the kinds of tasks being addressed. Engineering applications may not be generically different from the applications of expert systems in other areas, but the articles here present samples of what is stressed in engineering applications. For an introductory reading in expert systems, articles that appeared earlier in *Computer*[2-4] would make a good start.

The articles selected for this issue were chosen for the maturity of the applications they describe, their appropriateness (in addressing some of the technical issues of expert systems), and because as a group they present a balanced variety of applications and R&D activities. Many of these articles also demonstrate the need to integrate the knowledge-based part of the system with existing software and databases.

The first paper, by William Faught, discusses four different applications built on KEE. (KEE is a multi-paradigm programming environment that features frame-based knowledge representation.[5]) It emphasizes the importance of clear, visible models and advocates building a full system by starting out with a bag of tools or an intelligent workbench.

The next paper, by Giorgio Bruno, Antonio Elia, and Pietro Laface, describes production scheduling for flexible manufacturing systems, for which there is no known algorithmic solution. The authors make use of an intelligent, discrete-event simulation that is written partly in OPS5 rules and partly in Fortran code. (OPS5 is a popular data-driven inference engine.[6]) Although they note the flexibility and ability to incrementally develop the system that this approach makes possible, they report on the difficulty of maintaining data integrity between the two types of programs.

Gary Stroebel, Randy D. Baxter, and Michael J. Denney present a planning, design, and evaluation system for configuration of the IBM System/38 computer. This application makes use of many existing programs in the course of performing a variety of tasks in an interactive session. Performance modeling is in APL; workload analysis, user-interaction handling, and the configuration generator are in Pascal. An evaluation function has been implemented in three different versions: Lisp, OPS5, and ESE. (ESE uses the goal-driven inference in a manner similar to its use in Emycin, with an additional mechanism to specify certain controls.[7]) The global control and communication among these different modules is patterned after the blackboard model. The authors observe that it was convenient to use shells but add that they found them to be more awkward at times than straight Lisp, a case in point revealing the limited expressivity of knowledge representation in the current state of the art.

The blackboard architecture is intended for capturing and utilizing the control knowledge.[8] While this mechanism in its full generality allows flexible and explicit representation of high-level procedural knowledge, the overhead makes it impractical for performance-oriented applications. Many systems implement a simple form of such a mechanism.

Sarosh N. Talukdar, Eleri Cardozo, and Luiz V. Leão also make use of a simplified blackboard model to coordinate several different specialist modules in their system, Toast. The domain deals with monitoring and controlling utility power systems as an aid to human operators. The system has two parts: an off-line consultation part that has been implemented, and a real-time, on-line control part that has yet to be implemented. OPS5 has been adapted to run concurrently within their distributed problem-solving environment, called Cops.

Frank Pipitone reports on an expert diagnostic system for electronic circuits. The system has information on failure probability, a list of tests, and a list of costs. The system finds the "best next test" to apply, and upon receipt of the outcome, updates the "current resolution" listing. This system is unique in that explicit probability calculation is directly used with inferences made from component-wise behavioral rules and a component connectivity description. This system is written in Franz Lisp.

P. A. Subrahmanyam's Synapse system will be a large, complex system when completed. Intended for designing VLSI chips from specification to physical layout, the system is designed to deal with multilevel, multiperspective descriptions of the chip design in progress. Maintaining an algebraic model of the intended VLSI function, the system makes a series of refinement transformations on the model by use of production rules. This system also interfaces to many existing programs and to special-purpose hardware. The theorem-proving part is written in Pascal, the manipulation of algebraic models is done by Macsyma, algorithmic CAD work is done by a VLSI workstation, and the rule-based expertise is implemented with the KEE system.

The next two articles describe mechanical-design systems. David C. Brown and B. Chandrasekaran discuss a language designed to capture the task-level knowledge used in creating routine designs. "Routine design" refers to those design problems in which the intended function is well understood and the basic configuration of the design is known. (Nevertheless, the designer has to make design choices based on complex requirements, and then refine and specialize the chosen configuration.) This article represents a research effort to provide knowledge representation closely matching a particular kind of ex-

pertise, in this case that of routine mechanical design.

Sanjay Mittal, Clive L. Dym, and Mahesh Morjaria, on the other hand, attack nonroutine design—for paper-transport systems. Their article identifies, in terms of generate-test-analyze-modify cycles, necessary concepts for searching through a large design space. Their use of multiple specialists to handle design subgoals is similar to Brown and Chandrasekaran's. Their system already contains over 1000 objects implemented in Loops. (Loops is a multiparadigm programming environment that features object-oriented style.[9]) The authors also detail the experience in knowledge acquisition that they gained, which resulted in a 100-page document of paper-handling system expertise.

The last paper is by Yi-Tzuu Chien and Jay Liebowitz, who present a variety of defense-oriented application areas, along with system-level requirements. The authors point out a list of advances yet to be made before such expert systems can be practically implemented. These requirements and areas of necessary further progress are not unique to defense applications, nor to engineering applications. □

Acknowledgments

I would like to thank Mike Mulder, the editor-in-chief of *Computer,* and the designated member of his editorial board, the willing and able Ralph Preiss, who read all the manuscripts and with whom I consulted during the preparation of this special issue. I thank all the reviewers for their thoughtful reviews and all the authors who submitted their work for consideration in this issue. Eric Mays and Jim Griesmer helped me in selecting the reviewers and provided me with a number of helpful suggestions for this introduction. Last but not least, I would like to acknowledge the able secretarial help given me throughout the process by Evelyn Zoernack.

References

1. Donald A. Waterman, *A Guide to Expert Systems,* Addison-Wesley, Reading, Mass., 1985.
2. Dana S. Nau, "Expert Computer Systems," *Computer,* Feb. 1983, pp. 63-85.
3. *Computer,* special issue on knowledge representation, Oct. 1983.
4. F. Hayes-Roth, "Knowledge-Based Expert Systems: A Tutorial," *Computer,* Sept. 1984, pp. 11-28.
5. R. Fikes and T. Kehler, "The Role of Frame-Based Representation and Reasoning," *CACM,* Sept. 1985, pp. 904-920.
6. L. Brownston et al., *Programming Expert Systems in OPS5,* Addison-Wesley, Reading, Mass., 1985.
7. P. Hirsch et al., "Interfaces for Knowledge Base Builder's Control Knowledge and Application-Specific Procedures," *IBM J. R&D,* Vol. 30, No. 1, Jan. 1986, pp. 29-38.
8. B. Hayes-Roth, "A Blackboard Architecture for Control," *Artificial Intelligence,* Vol. 26, No. 3, July 1985, pp. 251-321.
9. Mark Stefik et al., "Knowledge Programming in LOOPS: Report on An Experimental Course," *AI Magazine,* Vol. 4, No. 3, Fall 1983, pp. 3-13.

Se June Hong received his BSc degree in electronic engineering from Seoul National University in 1965, and his MS and PhD degrees in electrical engineering from the University of Illinois in 1967 and 1969, respectively. He then joined IBM Poughkeepsie Laboratory, working in the areas of fault-tolerant computing and design automation. He joined IBM Thomas J. Watson Research Center at Yorktown Heights, New York in 1978. He is currently a senior manager in the Computing Technology Dept., responsible for projects in computer algebra, natural language, education, and knowledge-based programming.

During the academic year 1974-75, Hong was a visiting associate professor at the University of Illinois, Urbana. He was a visiting professor at the Korea Advanced Institute of Science and Technology (KAIST) for the month of October 1980.

He received an honorable mention award for Outstanding Young Electrical Engineer in 1975 and three outstanding innovations awards from IBM. He served as a Computer Society Distinguished Visitor, 1972-74; the guest editor of the special issue on reliable and fault-tolerant computing of *IEEE Transactions on Computers*, which appeared in July 1982; and as a member of the Ad Hoc Visiting Team for the IEEE Engineering Accreditation Board, 1979-84. He is currently a member of the Edison Medal Awards committee.

Hong is a fellow of IEEE and a member of ACM, MAA, ACL, AAAI, KSEA, and Sigma Xi.

Readers may write to Se June Hong about this special issue at IBM T. J. Watson Research Center (31-206), PO Box 218, Yorktown Heights, NY 10598.