EXPERT SYSTEMS

At a basic level of description, expert systems are computer programs. However, they are different from most other computer programs in several ways. Functionally, they differ because they can perform problem-solving or decision-making tasks in some well-defined domain at a performance level comparable to human experts. In terms of content, expert systems primarily encode and manipulate symbolic knowledge about some domain or some type of problem-solving task, rather than mathematical equations, algorithms, or numerical data. Because of their emphasis on knowledge rather than on numerical computation, expert systems are often known, more appropriately, as knowledge-based systems. Knowledge-based systems are characterized by their dependence on fairly specialized knowledge, of the kind that humans accumulate through experience, understanding, or insight. By contrast to mathematical models, which describe physical and chemical phenomena, knowledge-based systems are qualitative models that capture human decision-making processes (1).

From this follows the definitions:

Definition 1: An expert system is a computer program that manipulates large amounts of symbolic knowledge using qualitative techniques, to solve problems that can otherwise be solved only by expert human problem solvers. Expert systems capture the human problem solver's expertise in the form of domain-specific knowledge and domain-independent problem-solving strategies.

Definition 2: Knowledge-based systems are computer programs that encode symbolic knowledge about domains and tasks, and solve problems by manipulating this knowledge using qualitative techniques.

The term domain means some subject or area of expertise. The domain can be as broad as chemical engineering, or it can be a more specific subject such as biochemical processes, distillation columns, or catalytic reactors. A problem-solving task is an intelligent reasoning activity that humans perform well. Examples include selecting among several alternatives and deriving conclusions based on known facts. Knowledge, as far as expert systems are concerned, is an aggregate of facts, descriptions, and methods expressed in the form of symbols and qualitative relationships.

Definition 2 is phrased in terms of knowledge-based systems rather than expert systems. No reference is made to expert human problem solvers. Definition 2 captures the sense that the representation and manipulation of knowledge is the source of such a system's power, whether or not that knowledge is directly elicited from a human expert.

0.1. Importance of Expert Systems

One way to understand what expert systems are is to understand what they do. Mycin, one of the earliest expert systems, is a computer program designed as a decision aid for doctors (2). When given data describing a medical patient's symptoms, Mycin could diagnose infectious blood diseases and prescribe therapies appropriate to the disease diagnosed. Another early knowledge-based system, R1, is still used today by Digital Equipment Corp. (3). Given a set of specifications describing the type of computer system required by a customer, R1 selects the appropriate computer components and peripherals, checks for inconsistencies, designs the layout of the entire system, and prints out a detailed order. The order can then be used by a sales representative to deliver what the customer specified.

These two examples hint at a few of the reasons for the importance of knowledge-based systems. A medical facility may handle hundreds of infectious disease cases a year. Speedy, accurate diagnosis of these cases, aided by a system such as Mycin, may help the medical facility handle more patients, more effectively. Likewise, configuring large computer systems composed of many components can be a time-consuming and error-prone task. R1 accomplishes this task correctly in a fraction of the time it would take for human technicians. The savings in this case are in terms of the number of orders processed, which ultimately translates to dollars. As a final example, consider Prospector, another classic expert system built in the 1970s (4). This computer program, designed to detect commercially viable ore deposits based on geological data, correctly identified a molybdenum ore deposit worth about $100 million.

Other advantages of knowledge-based technology include the following: knowledge-based systems can automate complex tasks and do them quickly, even in fairly demanding domains, at a level comparable to their human counterparts; they can handle large amounts of data without suffering from information overload; they are tireless and potentially available round-the-clock; they are consistent, ie, if the inputs are the same, so are the outputs; they provide a readily available, lasting repository of expertise; and they have had proven successes in a variety of tasks, from diagnosis to design to data interpretation, and a variety of domains, from medicine to finance to chemical engineering.

Although knowledge-based systems have had many successes, there have been failures as well. Knowledge-based solutions may not be appropriate for certain problem-solving tasks. Alternative technologies may be more suitable, or the right approach to modeling the task may not yet have moved from the research lab to the shop floor. Failures have also happened for a variety of reasons associated with how the technology was applied and managed. Like some other technologies, expert systems have suffered from excessive hype and overblown expectations, both of which point to the need for care in appropriately applying the technology to the problem.

0.2. History

Knowledge-based system technology arose out of efforts to apply research in the field of artificial intelligence (AI) to practical problems. Broadly speaking, AI is concerned with trying to model intelligence and human problem-solving behavior using computational techniques. Central to this endeavor is the notion that computing machinery can emulate intelligence, a concept that was first formalized in 1950 (5). AI as a science is thus roughly 40 years old, and its early history makes fascinating reading (6). The evolution of AI can be divided into three segments, each characterized by certain key developments that led to knowledge-based systems:

The early period: In the 1950s and 1960s, AI was mostly concerned with developing computer programs that could perform tasks that were considered to require a high degree of intelligence, eg, game playing in domains such as chess and checkers (7); the solution of logic problems, such as theorem proving (8); and the solution of puzzle problems in toy domains (9). One key development of this period was the idea of heuristics, an important precursor to expert systems. Heuristics can be defined as guidelines for choosing among alternative courses of action. Heuristics can be used as shortcuts to direct the search for a solution along more promising lines, even if an optimal solution is not necessarily guaranteed. Another key development of this period was the creation of Lisp, a symbolic programming language.

The "understanding" period: The late 1960s and early 1970s saw AI begin to address broader aspects of intelligence. Research turned toward modeling cognition, interpreting natural language, story understanding, and ways to represent and reason about diverse kinds of knowledge (10–12). This period also saw the beginnings of applying AI research toward practical, real-world problems. Examples include Macsyma, which could solve mathematics problems symbolically (13); Strips, which could construct plans to achieve some goal, as an ordered series of steps (14); and Dendral, a program to assist in determining the molecular structure of chemical compounds (15). Programs such as these could perform at near expert levels in specialized domains, but still relied to some extent on search techniques rather than on a knowledge base.

The modern period: The significant breakthrough that led to the expert systems explosion of the late 1970s and early 1980s was the realization that computer programs could perform useful tasks at expert levels of performance, if they could be endowed with large amounts of specialized knowledge, and constrained to narrow but real domains. Instead of attempting to solve general problems of intelligence, research turned toward trying to clone human experts by capturing their experiential knowledge. Techniques were developed for extracting human expertise in the form of situation-specific rules, and encoding the rules into computer programs that could manipulate them and solve specific problems. Mycin, Prospector, R1, and other successful expert systems from this period were working proof of this concept. These successes led to the notion of an expert system shell, a skeleton program that had the basic structure in which rules about a new domain could be entered, and the basic pattern matching capability to make inferences using the rules. The simplicity of the shell concept, combined with the potential for wide applicability, resulted in rapid commercialization. Starting with the early 1980s, people started applying knowledge-based systems to a wide variety of domains, including chemical engineering.

0.3. Knowledge-Based Systems in Chemical Engineering

In the chemical engineering domain, knowledge-based technology has been applied mainly to the tasks of diagnosis and design. One of the earliest applications was Falcon (Faults AnaLysis CONsultant), built jointly by Du Pont, The University of Delaware, and Foxboro (16). This was a rule-based system interfaced to on-line process data, that could automatically diagnose malfunctions in an adipic acid reactor unit. By the standards of the day, Falcon was a relatively large knowledge-based application. Its early success, combined with its on-line implementation, sparked interest in knowledge-based technology in chemical engineering.

Examples of diagnostic applications include Diad-Kit/Boiler, an on-line expert system for performance monitoring and diagnosis of steam boilers (17), and CatCracker, a knowledge-based system for diagnosing operating problems in fluidized catalytic cracking units (18). Examples of design applications include BioSep Designer, which can automatically configure a process flow sheet and size the equipment involved in downstream separation of fermentation products (19); and Still, a tool for automated process and mechanical design of distillation columns (20).

Apart from design and diagnosis, other types of problem-solving tasks have also received attention in the chemical engineering domain, although not to the same degree. For example, knowledge-based reasoning techniques have been applied to simulation of discrete event processes (21), hazard and operability studies (22), intelligent multivariable process control (23), process operations planning (24), and scheduling of batch processes (25–27).

1. Technology Overview

From a technology perspective, knowledge-based systems (KBS) represent a new software methodology for solving certain types of problems effectively. It is important to understand what is encoded in knowledge-based systems, and how KBS technology differs from conventional numeric computational techniques.

1.1. Representation

From a software viewpoint, knowledge-based technology provides ways to represent knowledge and reason with it. The knowledge to be represented includes facts, descriptions, relationships, and problem-solving knowledge.

1.1.1. Facts About the Physical World

This category includes facts about specific objects or events, eg, reactor R-102 is refractory-lined, or the sun rose at 7:00 am yesterday. There may be generalized facts about the world, eg, grass is green, birds can fly, or copper sulfate crystals are blue. Assumptions on which these facts may rest, or conditions under which the facts no longer hold, are included in this category as well, eg, dry grass may be brown, or certain birds, such as penguins, cannot fly. Facts may also have associated with them the degree of certainty with which they are known.

1.1.2. Descriptions of Physical Objects, Processes, or Abstract Concepts

For example, pumps can be described as devices that move fluids. They have input and output ports, need a source of energy, and may have mechanical components such as impellers or pistons. Similarly, the process of flow can be described as a coherent movement of a liquid, gas, or collections of solid particles. Flow is characterized by direction and rate of movement (flow rate). An example of an abstract concept is chemical reaction, which can be described in terms of reactants and conditions. Descriptions such as these can be viewed as structured collections of atomic facts about some common entity. In cases where the descriptions are known to be partial or incomplete, the representation scheme has to be able to express the associated uncertainty.

1.1.3. Relationships Between Objects, Processes, and Events

Relationships can be causal, eg, if there is water in the reactor feed, then an explosion can take place. Relationships can also be structural, eg, a distillation tower is a vessel containing trays that have sieves in them; or relationships can be taxonomic, eg, a boiler is a type of heat exchanger. Knowledge in the form of relationships connects facts and descriptions that are already represented in some way in a system. Relational knowledge is also subject to uncertainty, especially in the case of causal relationships. The representation scheme has to be able to express this uncertainty in some way.

1.1.4. Problem-Solving Knowledge

The first three knowledge types are primarily declarative in nature, and most early approaches to knowledge-based systems generally focused only on these. In many cases, it is desirable to explicitly represent how-to knowledge, ie, knowledge about the reasoning strategies that are needed to manipulate the representations (28).

1.2. Reasoning

Knowledge-based systems need ways to manipulate their representations to produce useful results. Representation and reasoning are closely intertwined; how knowledge is represented influences and constrains the types of reasoning methods that can be brought to bear on the representations. Some fundamental techniques for reasoning are deduction, pattern matching, and search.

1.2.1. Deduction

If a knowledge-based system has a set of facts, and new facts are provided to it, then rules of inference can be applied to the set of facts to derive conclusions. For example, from the facts that (*1*) hydrogen and oxygen can react explosively at high temperatures, (*2*) air contains oxygen, (*3*) the atmosphere inside a particular reactor contains hydrogen, and (*4*) the reactor is at a high temperature, the additional fact that air has leaked into the feed to the reactor leads to the conclusion that an explosion can take place. The conclusion is based on applying deductive logic to the known facts. This is representative of the logic-based approach to knowledge-based systems.

1.2.2. Pattern Matching

If knowledge is expressed, not in terms of individual facts, but in terms of direct associations between situations and conclusions, then conclusions can be reached by trying to match the given situation with a stored conclusion. For example, consider a knowledge base with a number of associations relating types of service with the appropriate pump for that service. Assume that each association can be considered separately, ie, no interactions. Given a particular type of service, the task is to select the right pump. This selection can be made by trying to match the specifications of the current case with the preconditions of one or more associations in the knowledge base.

1.2.3. Search

In spite of the emphasis in KBS technology on knowledge being the source of problem-solving power, search is still one of the staple techniques used in manipulating representations. Search is closely associated with pattern matching capability. For example, if the pump selection knowledge base is very large, it would not be intelligent to try to match every association to the given specifications. Some means of constraining the pattern matching complexity would be required. A focused search, based on successively narrowing down the scope of the selection, would be one way to tame this complexity. Search techniques are also relevant in other tasks where the knowledge representation encodes solution alternatives or partial solutions.

1.3. Comparison to Conventional Numerical Programs

Fundamentally, knowledge-based systems manipulate symbols that have some meaning in the external world. For example, mathematical expressions, such as the one shown, can be integrated analytically, using the rules of integral calculus to manipulate the symbols representing the variables. This is how humans often solve integration problems.

$$\int \left(x^2 + 3x + 2\right) dx$$

By contrast, a numerical computer program for solving such integration problems would depend on approximating the mathematical expression by a series of algebraic equations over explicit integration limits.

Knowledge-based systems typically use qualitative methods rather than quantitative ones. For example, consider a simple tank system. The equation describing the flow rate of liquid out of the tank is given below, where C_{v} is the orifice coefficient, d is the diameter of the orifice, and h is the height of liquid in the tank. Based solely on the form of the equation, a human reasoner can infer that the flow rate F increases monotonically with the height h of liquid in the tank.

$$F = C_v \cdot d \cdot h^{1/2}$$

To reflect this type of reasoning, a KBS captures qualitative relationships between variables. By contrast, a conventional program that implements the flow equation calculates the value of the flow rate for numerical values of the input variables, ie, orifice diameter, orifice coefficient, and liquid height.

Humans deal robustly and efficiently with complexity by using shortcuts, guidelines based on past successes, and by relaxing the constraint of optimality. For example, in troubleshooting a process problem the human expert may use direct associations between causes and symptoms gathered from past experience. Similarly, knowledge-based systems make use of heuristic strategies for arriving at feasible solutions. There may even be more than one correct solution for the problem. By contrast, conventional numeric programs are usually oriented toward finding exact or optimal solutions. For example, an optimization program that calculates operating targets might try to find exact values for the decision variables, so that an objective function is maximized subject to algebraic constraints.

Human cognitive tasks also have to cope with uncertainty. For example, real world constraints are subject to change without notice, one's knowledge may apply only partially to a particular situation, and the data needed to infer conclusions may be incompletely known. Knowledge-based systems deal with such problems by explicitly representing and reasoning with uncertainty. By contrast, conventional numeric techniques are applied to well-defined problems that can be mathematically characterized. Uncertainty is implicitly dealt with by making simplifying assumptions, or using empirically determined model parameters. The key difference is that the computational process does not deal with the uncertainty; it is the developer of the computational algorithm who does. These remarks are applicable primarily to deterministic numeric models. Stochastic models do explicitly address uncertainty by incorporating random distributions.

The types of data structures and processing techniques used in knowledge-based systems are different from those used in conventional, numerical programs. For example, knowledge-based systems use data structures such as rules and objects and process these using pattern matching, inheritance, and message passing. Often, knowledge-based systems are data-driven, ie, processing is initiated in response to changes in data, not in a predetermined sequence. By contrast, conventional computer programs use data structures that are more suited for representing, storing, and manipulating numbers, eg, arrays, records, fields, and tables. The types of processing techniques used include arithmetic operations, logical comparisons, iterations, and relational calculus. In general, processing tends to be program-driven, ie, a sequence of instructions acts upon data in a predetermined fashion.

Knowledge-based systems differ from conventional programs in the types of applications for which they are used. Knowledge-based technology is most appropriate for implementing tasks that have some cognitive equivalent. Thus KBS applications automate tasks such as troubleshooting, conceptual design, or scheduling, whereas conventional programs address such number-oriented tasks as data storage and retrieval, equation-solving, optimization, and numerical simulation. This is not to imply that numerical programs are of no use in performing tasks such as design or scheduling; only that they are not well-suited to the cognitive component of these tasks.

2. Theory

From the viewpoint of AI and cognitive science, the theoretical question underlying knowledge-based technology is how to model expert problem-solving behavior. The predominant models of representation and reasoning in AI are logic, rules, and objects. More recent approaches to knowledge-based systems include task-specific models. The emphasis here is on applying the theory, ie, how do the problem-solving models and the representation concepts apply to real-world problems? The representation and reasoning techniques are described independently of any specific development tool, and illustrated using examples from chemical engineering wherever possible.

2.1. Logic

The rule-based representation used in many expert systems has its roots in formal logic. Logic programming has been applied successfully to build knowledge-based applications for certain problems (29, 30). It is a popular approach in Europe, and has been used in the much-touted Japanese Fifth Generation Computer Project. The logic approach supports a declarative style of representation; knowledge about a domain is represented in the form of facts or axioms. The fact base can then be queried about the truth or falsehood of some proposition to be tested. The logic-based system sets up the test proposition as a goal to prove, and uses rules of inference to do so. The most general of the inference rules is a method called resolution theorem proving (13).

Of the many variants of logic, one of the most important is first-order predicate logic. Facts in predicate logic are expressed using variables, constants, predicates, and quantifiers. For example, consider the facts shown below:

$$\text{For all } X \text{ such that Liquid } (X)\text{, Flow } (X) \qquad (1)$$

$$\text{Liquid (Water)} \qquad (2)$$

In statements 1 and 2, “For all” is a quantifier, specifically, the universal quantifier; Water is a constant; *X* is a variable; and Liquid and Flow are predicates. Both statements 1 and 2 are asserted to be true. First-order predicate logic gets its name because predicates such as Liquid and Flow are allowed, and quantification is only allowed on variables, not predicates. Having represented these facts in the logic system, the system can then be queried:

$$\text{Flow (Water)?} \qquad (3)$$

The system would respond:

$$\text{True} \qquad (4)$$

The logic system arrives at this answer by trying to prove the assertion in statement 3. The constant argument of the predicate Liquid is matched with any variable argument for the same predicate in other sentences in the fact base. The proof procedure is not as straightforward when the fact base contains a large number of sentences with different types of constructs. Sentences may have negations, eg, NOT(Corrosive(*X*)); conjunctions, eg, Corrosive(*X*) AND Viscous(*X*); disjunctions, eg, Corrosive(*X*) OR Viscous(*X*); existential quantifiers, eg, “For some *X* such that Acid(*X*), Corrosive(*X*).” In such cases, simple pattern matching and variable binding will not suffice; the entire resolution theorem proving method has to be applied. For details, the reader is referred to more comprehensive texts (13, 31).

First-order predicate logic is appealing from a mathematical standpoint because there are provable properties of logic-based systems (31). As a general methodology for implementing knowledge-based systems, however, first-order predicate logic has a number of practical shortcomings. There are restrictions on the expressiveness of facts because the syntax of assertions has to conform to the rigid framework of logic. There are no easy ways to structure or organize knowledge. Entities, events, processes, and the relationships between them have to be represented in a flat, one-dimensional space of facts. There are no straightforward ways to encode the uncertainty involved in assertions such as, “The presence of high concentrations of catalyst in the effluent may be indicative of poor settler operation.” Systems based on first-order predicate logic have difficulties dealing with time. Neither is there any way of retracting conclusions that no longer hold true, or specifying defaults. Last, but not least, logic-based systems are notationally complex to use.

Extensions of first-order predicate logic have been developed that address some of these shortcomings. For example, fuzzy logic, first introduced in 1965 (32), addresses the problem of representing and reasoning with uncertainty. A number of temporal logics have been devised for representing time (33). Default logics and nonmonotonic logics have also been devised for addressing the problems of representing defaults and retracted conclusions (34). Problems of usability have been addressed well in programming environments for building logic-based systems. However, the addition of extensions to first-order predicate logic takes away many of the provable properties of such systems. This partly undermines some of the motivations of elegance and mathematical provability that prompted the use of logic systems in the first place.

At the implementation level, the Prolog language is probably the most popular software for developing logic-based expert systems. In the chemical engineering domain, the logic-based approach has not been applied extensively. In the United States, rule-based and object-based systems have enjoyed much greater popularity. In Europe, despite the popularity of Prolog and logic-based approaches in the knowledge-based systems community, there are few accounts of implemented systems in the chemical engineering literature (22).

2.2. Rules

Rules, first pioneered by early applications such as Mycin and R1, are probably the most common form of representation used in knowledge-based systems. The basic idea of rule-based representation is simple. Pieces of knowledge are represented as IF–THEN rules. IF–THEN rules are essentially association pairs, specifying that IF certain preconditions are met, THEN certain fact(s) can be concluded. The preconditions are referred to as the left-hand side (LHS) of the rule, while the conclusions are referred to as the right-hand side (RHS). In simple rule-based systems, both the preconditions and conclusions are variable–value pairs. For example, a rule for diagnosing a reactor would be represented as follows.

IF reactor_temperature is high

THEN coolant_malfunction is established *Rule*

In this example, reactor_temperature and coolant_malfunction are variables, and high and established are values. Variables can take symbolic or continuous values. For example, the rule above can alternatively be stated as:

IF reactor_temperature is 150

THEN coolant_malfunction is established *Rule*

In general, both the preconditions and conclusions can have conjunctions (AND), disjunctions (OR), and negations (NOT). Conjunctions make a rule more specific, whereas disjunctions make a rule more general. A rule with a disjunction is equivalent to two separate rules with the same conclusion.

Figure 1 shows the architecture of a rule-based system. Fundamentally, there are three components: a rule base, a working memory, and an inference engine. The rule base contains discrete fragments of knowledge, all expressed as IF–THEN associations over variables. The working memory contains facts or assertions that are true at any stage in the computational process, ie, it contains variable–value binding. The inference engine is a domain and knowledge-independent processing mechanism. The inference engine searches the rule base, matching variable–value bindings in the working memory to preconditions and conclusions of rules in the rule base. When a match occurs, the rule is said to have fired. During this process, if a rule is traversed left to right, trying to infer conclusions from known conditions, the strategy is called forward chaining. If the rule is traversed right to left, trying to evaluate conclusions by checking if the appropriate conditions are met, the strategy is called backward chaining.

Forward chaining is useful when the objective is to find the implications of new pieces of information. This is applicable to tasks such as process monitoring, eg, determining the consequences of a change in process operating conditions, or design configuration, eg, determining how to sequence a series of separation steps. In forward chaining, the inference engine starts with some initial data, ie, values of variables in the working memory. It then fires every rule that applies to the starting data. The conclusions of the fired rules are also variable–value bindings that are introduced into working memory. The inference engine continues to apply rules based on the derived information until no more rules apply. An example of a forward chaining rule, used

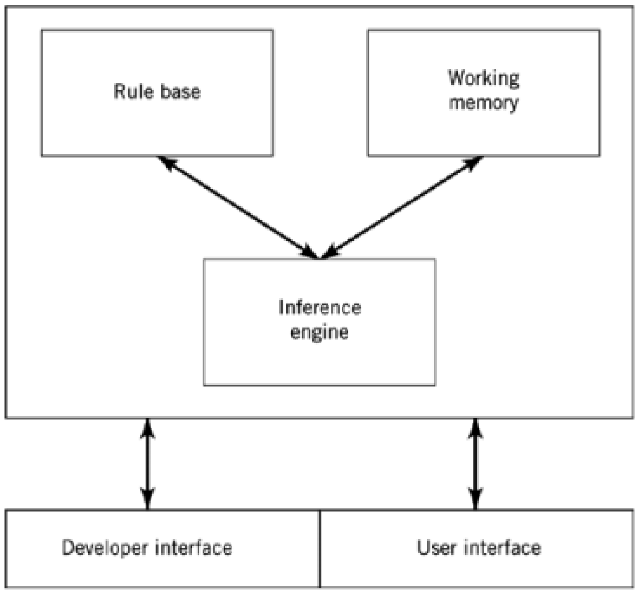

Fig. 1. Architecture of a rule-based system.

in a typical design configuration task, follows.

IF design_pressure is high

THEN configure safety_release_valve *Rule*

Backward chaining, on the other hand, is useful when the objective is to validate hypotheses. This is applicable to diagnostic and troubleshooting tasks, eg, finding the root cause of a process problem, or equipment selection tasks, eg, selecting a pump for a particular type of fluid service. In backward chaining, processing starts with the introduction of a goal into the working memory, ie, a conclusion to test. The inference engine tries to find all rules with the goal in their conclusions. Each rule that fires results in the precondition clause being added to the working memory as another goal. If data in the working memory can be bound to a goal, then that helps to validate or negate the goal. This process continues until no more rules fire. The reactor coolant malfunction rule (see Rule 1) is an example of a backward chaining rule used in a diagnostic task. The precondition of the rule is a symptom of a malfunction, while the conclusion represents a potential root cause. The objective of the system is to evaluate plausible conclusions by checking their preconditions against known data.

Forward and backward chaining can be combined in the same rule-based system to address subtasks in the overall problem. For example, the monitoring component of an on-line troubleshooting system can be designed to work in a forward chaining mode, while the component that determines root causes can be designed using backward chaining. When more than one rule is applicable in a given situation, the inference engine uses conflict resolution to decide which to fire. Several conflict resolution strategies can be used: (*1*) apply rules in the order in which they appear in the rule base (this is the most simplistic); (*2*) apply the most specific rule first; (*3*) apply rules according to a priority assigned at development time (static priority); and (*4*) apply rules according to a priority computed at run-time, based on other variables (dynamic priority).

Rules may represent either guidelines based on experience, or compact descriptions of events, processes, and behaviors with the details and assumptions omitted. In either case, there is a degree of uncertainty associated with the application of the rule to a given situation. Rule-based systems allow for explicit ways of representing and dealing with uncertainty. This includes the representation of the uncertainty of individual rules, as well as the computation of the uncertainty of a final conclusion based on the uncertainty of individual rules, and uncertainty in the data. There are numerous approaches to uncertainty within the rule-based paradigm (2, 35, 36). One of these approaches is based on what are called certainty factors. In this approach, a certainty factor (CF) can be associated with variable–value pairs, and with individual rules. The certainty of conclusions is then computed based on the CF of the preconditions and the CF for the rule. For example, consider the following example.

$$\text{reactor\_temperature is high with CF} = 80 \qquad (5)$$

$$\begin{aligned} &\text{IF} \quad \text{reactor\_temperature is high} \\ &\text{THEN coolant\_malfunction is established} \qquad (6) \\ &\text{Rule CF} = 90 \end{aligned}$$

Then the certainty of the conclusion, namely "coolant_malfunction is established" is computed as follows:

$$\begin{aligned} \text{CF (conclusion)} &= \text{CF (precondition)} * \text{CF (Rule)}/100 \\ &= 80^{*}90/100 \qquad (7) \\ &= 72 \end{aligned}$$

Thus certainty factors express the intuitive notion that the certainty of a conclusion should be lower than the certainty of the data and knowledge involved in arriving at the conclusion. Certainty factor theory also allows for combining the CFs of conjunctions, disjunctions, and negations:

$$\text{CF (NOT (A))} = 100 - \text{CF (A)} \qquad (8)$$

$$\text{CF (A AND B)} = \text{minimum}\left[\text{CF (A)},\ \text{CF (B)}\right] \qquad (9)$$

$$\text{CF (A OR B)} = \text{maximum}\left[\text{CF (A)},\ \text{CF (B)}\right] \qquad (10)$$

The certainty factor approach has been among the more popular rule-based approaches to uncertainty. However, although it is easy to apply given the individual CFs, acquiring the raw CFs from the experts is often quite difficult. Further, although the formulas for CF combination are mathematically appealing, they often have no relation to the ways in which experts combine evidence to arrive at conclusions. Some of the task-specific approaches discussed later address uncertainty combination in a more intuitive way (35).

Rule-based systems address the issue of explaining conclusions as well. During the inference engine's processing, two types of explanatory questions can be answered: why and how. Why questions arise when the system makes requests for data in order to validate hypotheses. The system responds to this type of question by referencing the rule which has the data under question, eg, "The reactor_temperature is required because Rule 102 requires it to validate coolant_malfunction." Similarly, a rule-based system responds to the how question by tracing through all the rules that led to the final conclusion, and referencing them. In addition, rules can be associated with explanatory text, which the inference engine can retrieve to provide clearer explanations, eg, "The reactor_temperature is required because a high reactor_temperature may be indicative of a possible coolant_malfunction, which is the hypothesis currently under consideration in Rule 102. The knowledge for this rule was obtained from Jane Q. Expert."

Rules are clearly a useful form of representation for knowledge-based applications, with their advantages of representational simplicity, wide applicability, and history of past successes. However, certain important design criteria govern the proper application of rules and there are shortcomings of the rule-based representation.

2.2.1. Design Criteria

In any but the most simple systems, rule organization is important for problem-solving and system maintenance. Although individual rules may be robust, they may interact in strange and unexpected ways with other rules, if they are put into a large unstructured rule base. Grouping rules according to a common set of goals or contexts is important for this reason. For instance, an on-line troubleshooting system may have an overall rule organization based on the following subtasks: monitor the process, detect abnormalities, find the root cause, and recommend corrective actions. As far as possible, conflict resolution should be minimized, again by rule organization. The basic capabilities of the inference engine, ie, forward or backward chaining, may not be sufficient to capture the problem-solving strategy. Some higher level control of the problem-solving process, representative of the task being done, may be required. Such additional mechanisms have to be encoded as strategy rules, called meta-rules in the literature (37). Meta-rules control the application of the knowledge rules, or set the context in which the knowledge rules should be applied.

2.2.2. Shortcomings

Rule representations based solely on variable–value bindings are restrictive in their scope. For example, consider a knowledge base that contains rules for selecting rotary pumps for various fluid services. If it later turns out that some of the rules are also applicable to selecting positive displacement pumps, the original rule set must be modified to include positive displacement pumps using OR operators in the conclusions of those rules. Thus there is no easy way to generalize over classes of pumps. Secondly, there are other types of relationships essential to capturing knowledge about a domain that are outside the scope of a rule-based representation. Examples are "type-of" or "part-of" relations. The use of meta-rules, unavoidable in many cases, has the disadvantage of obscuring the actual problem-solving strategy used by the system. This is a problem for the developer. It is also a problem for the end user who is confronted with explanations that are far removed from the actual reasoning process. Providing clear explanations of problem-solving is a research area in itself (38, 39).

2.3. Object-Oriented Representation

Object-oriented representation addresses many of the shortcomings of rules, especially when used in combination with a rule-based representation. In contrast to the type of data structures used in traditional programming, eg, variables, records, or linked lists, the basic data structure in object-oriented systems is an object with associated properties. For example, a process temperature measurement for reactor R-101 would be represented by the object R101_TEMPERATURE, with properties RAW_VALUE, SYMBOLIC_VALUE, and SYMBOLIC_TREND. Properties of an object can take values, eg, the SYMBOLIC_VALUE property of R101_TEMPERATURE might take the value HIGH. In many object-oriented systems, the object–property value holder is referred to as a slot, eg, R101_TEMPERATURE.SYMBOLIC_VALUE is a slot with a value of HIGH. Objects that share properties can be generalized into a class. For example, if there are multiple reactors, all reactor temperature objects could be grouped into a class called REACTOR_TEMPERATURES. In this sense, an object is also referred to as an instance of a class. Similar classes in turn can be grouped into higher level classes, ie, the REACTOR_TEMPERATURES class and other temperature measurement classes

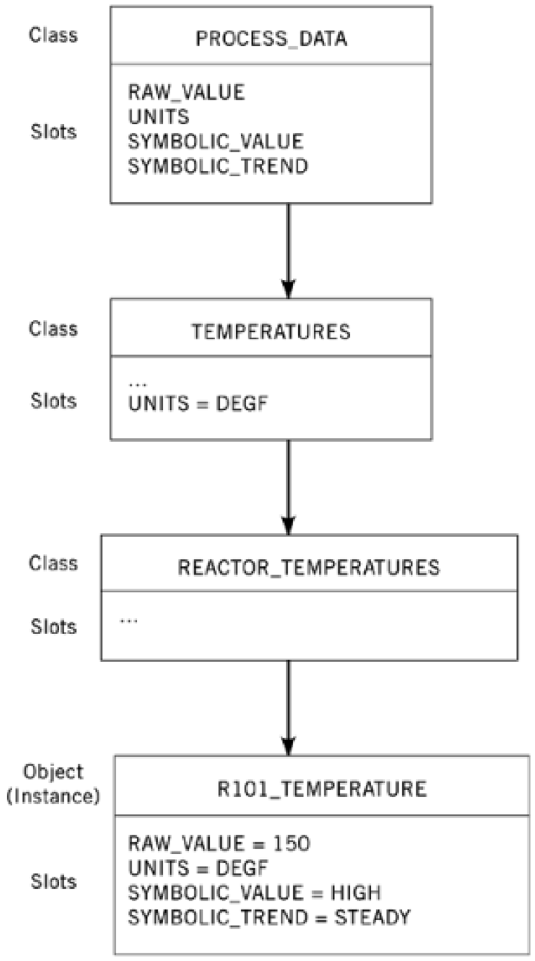

Fig. 2. Object-oriented representation.

could be grouped together under TEMPERATURES. In turn, this and other classes, such as PRESSURES, can be grouped under the class PROCESS_DATA (Fig. 2).

Some object-oriented systems also support the notion of subobjects. This facilitates the representation of structural relationships. For example, the objects IMPELLER and LINING can be made subobjects of a REACTOR object, to represent the structural components of the reactor.

To illustrate the power of the object-oriented style of representation, consider the reactor diagnosis example used earlier in the discussion of rules. Assume that there are several reactors, R-101, R-102, etc, each served by a common cooling system. Relating coolant malfunctions to the temperature in each reactor would need multiple rules, or rules with multiple disjunctions. Instead, if rules are used in combination with the object representation described above, a single general rule can be written to cover all the specific instances, as follows.

```
IF    REACTOR_TEMPERATURES.SYMBOLIC_VALUE is HIGH
         THEN coolant_malfunction is established         Rule
```

In object-oriented systems, the notion of inheritance is important. Inheritance means that a property of a class is automatically acquired by subclasses and objects of that class. For instance, having defined all the classes and objects listed above, if the property TIMESTAMP has to be added to all process data objects, one simply has to add it to the class PROCESS_DATA. This modification would be automatically reflected in all subclasses and objects.

Another important idea in object-oriented systems is being able to specify how slots can obtain values and what types of values they can hold. This allows specification of data types for slots, initial and default values, and restrictions on value ranges. Also, procedures to execute for obtaining the value can be specified, or even procedures to execute after getting the value. These types of procedures are referred to as daemons. For example, in the objects belonging to the REACTOR_TEMPERATURE class, the SYMBOLIC_VALUE slot may change, so it would be convenient to call a procedure to calculate the symbolic value on an as-needed basis. A daemon can be placed in the slot to do this, based on current values in the RAW_VALUE slot. The RAW_VALUE slot can also be restricted to a certain range of values if it is known that reactor temperatures cannot physically exceed some upper bound. Daemons, and other information describing the values a slot may take, are entered into what is termed the facet of the slot.

These discussions address the issue of representation using objects, but what about reasoning? One approach is to couple the object representation with a rule-based system and let the associated inference engine take care of the reasoning. This is the technique used in a number of knowledge-based applications, and it successfully addresses many limitations of pure rule-based systems, eg, the reactor diagnosis example. However, some of the other restrictions of rule-based systems, namely limited problem-solving capabilities and the inadequacy of meta-rules, still remain. To address these, object-oriented systems offer powerful capabilities for implementing many different reasoning strategies, via encapsulation and message passing. Encapsulation allows procedures and data to be bundled together into objects. Rather than data being passive, and acted upon by procedures, encapsulation allows data to be active, and procedures to work differently depending on the object they are attached to. Daemons are an example of this. The second feature, message passing, allows objects to communicate. The overall reasoning process is accomplished as follows: objects send messages to other objects; the recipient objects respond by executing their associated procedures; results or values are returned to the sending objects. Using these advanced features of object systems requires the use of an integrated programming language such as Lisp or C. These types of capabilities are somewhat on the fringes of mainstream knowledge-based systems technology. They are more applicable in the area of model-based systems.

One of the many advantages of an object-oriented representation is the fact that it allows representation of all the declarative information known about a process or domain in a structured way. Representing the domain knowledge declaratively offers the flexibility to choose whatever reasoning strategy is appropriate for the problem to be solved. In addition, the idea of inheritance greatly facilitates building and maintaining systems using an object-oriented representation. On the downside, object systems do not generally come with built-in problem-solving strategies, uncertainty handling capability, or explanation features. This is up to the system developer to implement using a programming language. For a range of knowledge-based applications, this potential hurdle can be avoided by coupling the object system with a rule base. Many commercially available tools support this hybrid architecture. For other types of problems, the hybrid approach may not be sufficient, and the full power of object-oriented programming may have to be used. For example, knowledge-based simulation of discrete event processes can be approached in an object-oriented way (21).

2.4. Advanced Topics

2.4.1. Model-Based Reasoning

Much of the knowledge in mainstream knowledge-based systems is associational in nature. Causal relationships, for example, are encoded as direct associations between symptoms and root causes, suppressing the details of any intermediate causal links. Knowledge in this form is also referred to as compiled knowledge (40). The advantage of this approach is that at problem-solving time, the knowledge is available to the system in the form needed. No explicit reasoning is required to derive the association. The downside is that if the knowledge-based system runs into a situation where those details become important, then it may fail. This is the brittleness problem. One way to address the brittleness problem within the framework of mainstream knowledge-based systems, is to structure knowledge at successive levels of detail, so that additional detail is available when needed (18). Nevertheless, the details are still represented in compiled form, ie, they are not derived at problem-solving time. Ensuring that all the detail is represented explicitly as associations places some burden on the developer, and there are limitations on how much detail can be explicitly represented.

Model-based reasoning attempts to address the brittleness problem in a more fundamental way, by modeling the behavior of the important entities in the domain. The objective of this modeling effort is to enable the reasoning system to derive any associational knowledge at problem-solving time in a flexible way. For example, if the task is troubleshooting a chemical process, the fundamental entities are the process equipment and the unit operations that take place within them. The goal would be to qualitatively model both the normal and malfunctioning behaviors of the process equipment using first principles. Modeling reduces the burden of having to explicitly represent the behaviors of the entities in a compiled form, but requires additional development time. The advantage is that the modeling effort can be leveraged for future applications, eg, if generic models are developed for process equipment, then they can be re-used for other processes.

In general, the model-based reasoning approach is best applied, not as a method in itself, but as an add-on to a knowledge-based system. The main reason is that modeling is hard, and problem-solving based solely on fundamental models is computationally complex (41). Using the hybrid approach can take advantage of the efficiency of compiled knowledge in rapidly focusing on the solution, while retaining the robustness of models when confronted with the need for behavioral detail. Several recent research papers in the chemical engineering literature have explored this hybrid problem-solving notion (42, 43). For more information on model-based reasoning in general see References 44 and 45.

2.4.2. Task-Specific Problem Solving

Rules and objects are examples of general problem-solving approaches, ie, the representations and reasoning constructs can be applied to different tasks. The work of mapping the needs of the task to the constructs available is left to the developer of the knowledge-based system. This can be a difficult knowledge engineering problem. For example, in the design configuration task, the constructs of importance are the components (or devices) to be included in the final configuration, the functions that those components can satisfy, and the design specifications to be met by the configuration. To build a knowledge-based configuration system, the developer has to map these task-level concepts to the basic constructs provided by rules, objects, or logic. Issues of how to appropriately represent the task-level knowledge, and how to structure the design rules in the rule base, for example, have to be resolved. Thus rules and objects can be regarded as fine-grained approaches to representation and reasoning, in the sense that they deal with problem-solving at a low level.

Task-specific problem-solving approaches address these issues of knowledge engineering, representational ease, and representational adequacy (28, 46). Such approaches are oriented toward characterizing the representation and reasoning needs of different cognitive tasks in a generic way. The advantage is that if a knowledge-based system has to be constructed for such a task, the developer has a ready-made blueprint to work with. The blueprint specifies what representational constructs are required, what reasoning strategies are appropriate, what knowledge is required to build the application, and how to go about acquiring the knowledge. Some researchers in this area have also focused on building tools specific to the tasks, so that even the representation and reasoning methods are available to the developer in a prepackaged form (28).

Task-specific approaches have a number of advantages, eg, knowledge structuring to overcome brittleness, ease of development, knowledge acquisition, reusability, and ease of explanation. From a developer's point of view, the downside is that the problems to be solved have to be appropriately matched to known tasks, and if not, a new task characterization has to be done. From a research point of view, there are issues of granularity and knowledge sharing to be addressed: at what level of detail should the task be characterized? How can the same fundamental knowledge be applied to different tasks? Nevertheless, task-specific approaches have been successful in guiding knowledge-based system development. Several large-scale problems in the chemical engineering domain have been solved using this approach (18, 20).

2.4.3. Real-Time Systems

Reasoning with time and having to solve problems under time constraints place unique burdens on a problem-solving system. To address these, knowledge-based systems need additional capabilities: problem-solving strategies to deal with the effect of time, increased processing speed, and enhanced integration capabilities, to name a few. Although real-time often means different things to different people, there is general agreement among researchers on the types of issues that real-time systems need to address (47, 48). These are as follows.

speed: the system must be capable of completing its computations faster than the rate of arrival of fresh information

responsiveness: the system should be able to recognize and respond to events in the time scale of interest

pre-emption: the system must be capable of pre-empting ongoing tasks when interrupted by unscheduled events of greater importance

adaptability: the system must be able to change its behavior over time

temporality: the system must be able to represent time and comprehend temporal relationships among events

Of these, the first two place demands on the processing speed of the knowledge-based system and its integration with the environment, eg, data acquisition interfaces. The third impacts on the nature of the computational cycle and the system's ability to reschedule its own computations. The latter two place demands on the representation and reasoning capabilities of the system, and are the most interesting from a knowledge-based system viewpoint.

Temporal reasoning is nonmonotonic in nature, ie, conclusions that are made in a certain time step may have to be retracted in future time steps. Conclusions may have persistence, ie, once made, they may last for a certain period of time before being subject to change. Two concepts are of particular importance in temporal reasoning: state and event. A state is associated with some aspect or condition of the physical world, and may change with time within the time scale of interest. For example, high reactor temperature is a state of the process. It may change over time. On the other hand, an event is associated with a unique point in time. For example, a reactor runaway took place at 7:00 am on 7/2/89 is an event, unchanging with time. In most systems that do not operate in real time, states are of primary importance to the task, eg, the reactor diagnosis example used earlier in this article. In real-time knowledge-based systems, sequences of events provide additional evidence for making conclusions. All of these additional representational capabilities are needed for developing real-time knowledge-based systems (49). The representation and reasoning needs can either be addressed by making use of the rich, extensible features of object-oriented programming, or by extending the capabilities of rule-based systems. Some development tools provide built-in constructs to handle some of these representational needs, eg, G2 from Gensym Corp. There are few detailed accounts of implemented real-time knowledge-based systems in the chemical engineering domain. The Falcon system mentioned earlier is one example (16), but there are others (50–52).

3. Building Applications

The purpose of this section is to provide some understanding of where and how to appropriately apply knowledge-based technology, and to give an industrial perspective on the process of developing and delivering knowledge-based applications. The literature contains numerous detailed discussions concerning knowledge-based system development (53–55).

3.1. Technical and Business Issues

In general, the successful application of any technology is dependent on both technical and business issues. For knowledge-based systems, six critical issues are problem identification, user acceptance, measurement of impact, appropriateness, feasibility, and cost.

There should be an identifiable problem that requires a knowledge-based solution. Typically, problems amenable to a knowledge-based solution are identified by the existence of a knowledge bottleneck, ie, whenever the successful execution of a task is delayed, limited, or compromised by the speed or availability of an expert. For example, when process upsets occur during night shifts, the rapid resolution of the problem is hampered by the unavailability of an experienced process engineer. Alternatively, the rapid evaluation of a large number of design choices is often limited by the processing speed of the design team. A third type of identifiable problem is the routine task, which if automated would free up the human experts to expend their energies in more creative activity. An example of this is equipment selection, the automation of which would permit the design engineer to spend more time creatively exploring the design of the process flow sheet.

The importance of considering the potential user of the application cannot be emphasized enough. This is particularly true of knowledge-based systems because they usually involve some degree of human interaction. Most knowledge-based applications to date have been designed to serve in an advisory capacity, keeping the human decision maker in the loop. For example, even if the technical or operating departments at a manufacturing facility identify a potential knowledge-based system application, the ultimate user of the application has to endorse and accept the application and participate in its development.

The impact of a knowledge-based application may appear in many ways: improved competitive position, quality improvement, improvement in efficiency, cost reduction, and reduction in downtime, to name a few. Some of these benefits may be hard to quantify; others may not be quantifiable at all. For example, the actual benefit derived from a diagnostic advisory system may not be apparent if the process behaves normally. To quantify the benefits, a careful post-audit may have to be done, taking into account the number of faults averted, and comparing the frequency of faults before and after implementation.

The appropriateness (or otherwise) of knowledge-based technology may become apparent during problem identification. In general, knowledge-based systems are appropriate when the task involves some symbolic reasoning, the task uses at least some symbolic information, the domain of the task is narrow, there are human experts who can perform the task successfully in a few hours or a few days, and infallible performance is not a requirement. Knowledge-based systems, unless designed for graceful failure or augmented by other robust AI techniques, can potentially fail in a brittle fashion if completely novel situations are encountered or if the inputs are noisy and incomplete.

Alternative technologies should also be considered, and the reasons for not using them should be justifiable. For example, database technology is not the right choice if a task requires reasoning that goes beyond retrieval of stored data based on well-defined criteria. At the same time, many problems that are stated in a symbolic way can be formulated mathematically, and in fact can be better solved numerically. For such problems, knowledge-based systems are not the appropriate answer.

Even if knowledge-based systems are the correct choice, there may be potential roadblocks to successful implementation. For example, experts should be available and willing to be debriefed. There should be adequate support from management. End users should participate in the effort. There are technical issues such as the selection of the right tools for implementation. If knowledge-based technology is just one component of the overall solution (as it often is), other technologies which must be integrated should receive careful consideration. The required expertise for implementing knowledge-based solutions must also be available.

Most of the cost of building knowledge-based systems is associated with the time and effort involved in knowledge engineering, which is often an iterative exercise. Probably the best way to estimate this cost is by using past experience with similar projects as a basis. Common development times range from one to six work months for small systems, six months to a year for medium-sized systems, and one to two years for large systems. One approach to cut down this development time is by building knowledge bases in a reusable way. Apart from the cost of knowledge-based development, there may be development costs associated with integrating the knowledge-based system with other systems if needed, the cost of delivering and installing the application, and training and documentation. Other costs such as the license and annual maintenance cost of software used should also be factored in.

3.2. Approaches to Development

In general, there are two approaches to knowledge-based systems development: the do-it-yourself approach, and the technology approach. In the do-it-yourself approach, the experts, who are the knowledge sources, learn enough about the technology and the tools to construct their own knowledge-based systems. This approach is successful when applied to small, well-defined problems, but education in the technology is a critical prerequisite. The do-it-yourself approach is also well-suited to highly decentralized organizations, where the burden of development and deployment is often on local groups of users.

The technology approach is one in which knowledge-based system professionals interact with experts and end users to develop applications. This is often the only option if the application involves even moderate complexity, and low end expert system shells prove to be inadequate to represent the knowledge or the reasoning. Irrespective of the approach taken, the development process can be roughly broken down into the eight steps shown in Figure 3. The type of approach taken simply changes the relative importance of the steps. The only exception is that in the do-it-yourself approach, education may be a prelude to the eight steps discussed here. The experts or end users may have to undergo some training in the theory and application of knowledge-based technology, and the mechanics of using specific expert system tools.

3.2.1. Problem Selection

To select the problem correctly, the criteria discussed earlier should be carefully applied before launching a project: the existence of a knowledge bottleneck, the inapplicability of exact numerical methods, the existence of either an expert or a theory for the task, the narrowness of the domain, and the business issues of payout and cost. If needed, the various criteria can be quantified and weighted based on their importance to the problem in order to rate potential projects (56). If the application of knowledge-based technology is either purely exploratory or educational, this step may be dispensed with. The problem selection step will likely involve the end users, the experts, and management.

3.2.2. Problem Analysis

The characteristics of the problem affect the knowledge representation and reasoning requirements, which in turn impact on the tool used for development. Analyzing the characteristics of the task and the types of knowledge available can help identify these requirements. In many cases, this can be accomplished by protocol analysis, ie, a detailed step-by-step examination of actual problem-solving cases performed interactively with the experts(s). For example, the protocol analysis may reveal that the task consists of a well-defined sequence of decisions, with context-specific knowledge being brought to bear at each decision step. This may suggest a decision tree representation, with rules to evaluate each decision node. If the knowledge is available either as a collection of discrete facts or in the form of guidelines for actions, a rule-based or logic-based representation

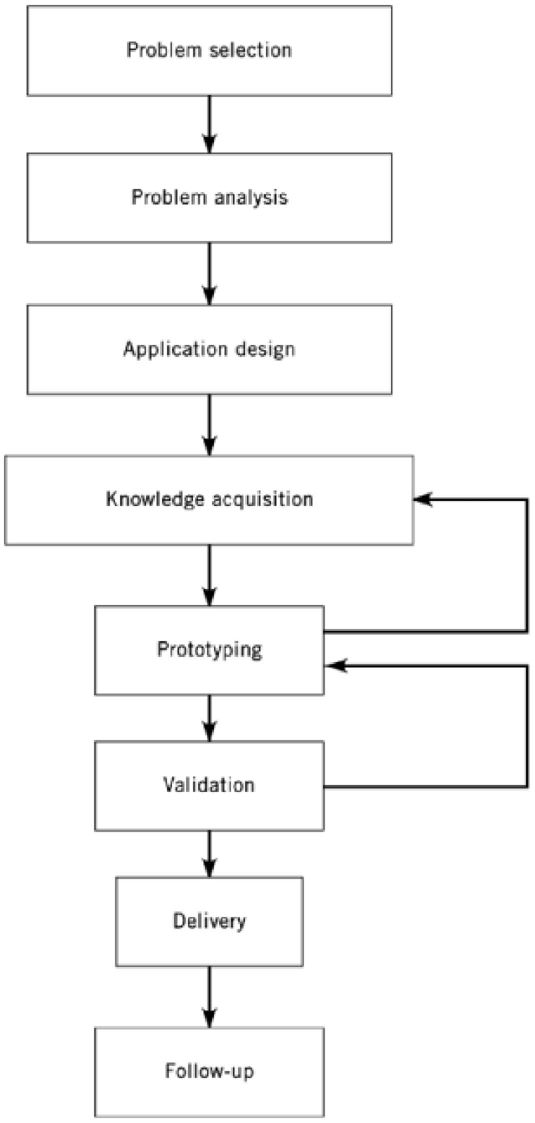

Fig. 3. Steps in the development process.

may be the appropriate choice. If the knowledge used in the task consists of relations such as "type of" or "part of," or if the rules can be generalized by applying them to classes of objects, then an object-oriented representation may be useful.

It is also useful to try and classify the problem into broad categories, such as diagnosis, design, selection, planning, or scheduling. This may help to map the problem directly into a predefined representation/reasoning schema (57). If the task requires finding the best of many alternatives, then it is likely to involve search strategies. If the execution of the task is likely to involve many novel situations not generalizable a priori, then model-based reasoning or first principles reasoning may be called for. In many cases, there may be several components in the overall task. For example, diagnosis may involve data abstraction (interpretation

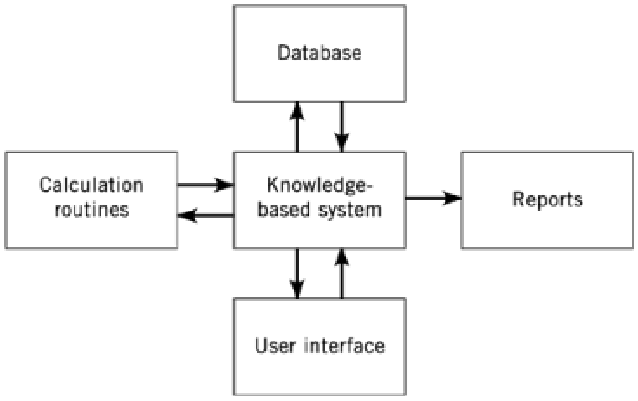

Fig. 4. Architecture of a typical integrated application.

of raw data), determining root causes, and selection of remedial actions (58). Problem analysis should be kept independent of domain details as far as possible.

3.2.3. Application Design

Application design involves fleshing out the functionality of the overall system. Rarely, if ever, is knowledge-based technology used in isolation in applications. The developer has to consider the integration of the knowledge-based component of the application with other components such as the user interface, data acquisition interface, and access to numerical routines for calculated data, as required. Figure 4 shows a typical application architecture that incorporates these components. End users and information systems staff are likely to be heavily involved in the application design phase, at least in a consulting capacity.

3.2.4. Knowledge Acquisition

The process of knowledge acquisition is dependent on the results of problem analysis, ie, the representation and reasoning methods chosen for the problem guide the knowledge engineer through the elicitation (59). For example, knowledge acquisition for a diagnostic application would focus on the various root causes relevant to the domain, associations between root causes and symptoms, and ways to organize the space of root causes to better focus the search. However, there are a few general methods that are useful, independent of the task. These include protocol analysis, repertory grids, and induction from examples (36). Protocol analysis is a technique for obtaining case studies and following the detailed steps in the expert's chain of reasoning. Repertory grids are a way of organizing associations in the form of tables, noting differences between associations, or other attributes of the associations, eg, level of confidence. From an object-oriented point of view, repertory grids are particularly useful for identifying features of interest, and for assigning values to those features. Lastly, induction from examples is a useful approach when the representation is simple and relatively unstructured. Given actual problem-solving examples of data and the conclusions resulting from the data, an induction system can generalize a decision tree or a set of unstructured rules. Knowledge engineers and experts are the primary participants in the knowledge acquisition step.

3.2.5. Prototyping

As Figure 3 shows, the prototyping step is a parallel activity with knowledge acquisition, conducted in an iterative fashion. Unlike traditional software applications, knowledge-based systems are difficult to specify exactly during the problem analysis phase. Intermediate results from the prototyping step may help with further knowledge acquisition. The prototypes can help get feedback from the end users. Prototyping also results in a phased development effort; as modules of the system get developed, they can be tested and even used, while development on other modules progresses. This step requires a good understanding of representation techniques, reasoning methods, and the features of the development tool. Experts and knowledge engineers are the primary participants in this phase; however, end users may also be involved to provide feedback on system performance.

3.2.6. Validation

Validation by experts and end users drives the iterative process shown in Figure 3 for prototyping and knowledge acquisition. Validation is also concerned with how to test the system after the knowledge engineers and experts have completed their work and produced a fully functional application. This is an important but difficult activity. The performance of the system has to be measured, its robustness evaluated, and its areas of brittleness identified. Performance measurement, as used here, refers to measuring the accuracy of the knowledge-based system's conclusions, and the confidence level to be expected from the system. Other common performance measures include execution speed and response time. An obvious performance measurement is to test the application on available case studies, for which the inputs and results are known. However, this should not be the only method of validation used because available cases usually constitute a small fraction of the scope of the system. There are other validation methods that focus on the content and consistency of the knowledge in the system, checking for rule conflicts, redundancies, circular rules, and knowledge gaps (60).

3.2.7. Delivery

This includes installation and commissioning of the application, user training, manuals, and system documentation. Delivering knowledge-based systems is not very different from delivering other types of applications, with the exception that end users may need more time and assistance in getting familiar with using the system. It is important to keep the needs of delivery in mind during the early phases of the project, since they can have a profound influence on overall system design (not just the knowledge-based component), and can substantially impact time and cost estimates for projects.

3.2.8. Follow-Up

Even after the system is delivered, additional work is usually needed. In conventional software engineering this is viewed as software maintenance or upgrade. It is referred to here as follow-up to emphasize that the process of knowledge-base refinement that begins in the prototyping stage can continue even after the system is commissioned. The need for additional detail may come to light only after the application has been in use for some period of time. The time required to implement such enhancements should be taken into account.

3.3. Implementation Tools

In the mid to late 1980s, the expert system tools market was flooded with a plethora of expert system shells of every imaginable flavor, all varying widely in their representational and development capabilities. The 1990s have seen the market narrowing considerably; many of the tools have converged on a common set of features. Nevertheless, choosing an appropriate tool for knowledge-based application development is not easy. A number of knowledge level criteria have to be considered, including representational capabilities, eg, rules and objects; knowledge organization capabilities, eg, rule sets and taxonomies; reasoning capabilities, eg, backward and forward chaining, rule prioritization; uncertainty handling features, eg, certainty factors and belief values; truth maintenance or assumption maintenance capabilities; and special-purpose features such as temporal reasoning capabilities, which may be needed for real-time applications, or planning and scheduling systems.

In addition, other capabilities relevant to system development, end user interaction, and application integration have to be taken into account: development features, eg, graphic displays of knowledge organization, knowledge-base maintenance features, explanation facilities, system documentation capability, end user interface development tools, availability of an integrated programming language for custom coding, access to external data files, interface to external programs, hardware platforms supported, vendor reliability and service, and price.

In selecting an appropriate tool, two other factors may also be important. First, it may be beneficial to leverage higher initial cost for a more full featured tool, if future applications are likely to need those capabilities. Second, more than one tool may be necessary to serve the needs of different types of applications, ie, the adage "one size does not fit all" holds well in the tool business. Available tools in the market can be categorized into roughly three classes.

3.3.1. Low End Tools

These are suitable for learning about the technology and building small applications, but have limited representational capabilities. Typically, the tools in this category are those that a domain expert, not knowledgeable in expert systems technology, might use. Low end tools are also suitable for providing hands-on training to people new to the technology. Some examples of commercially available tools in this category are VP-Expert (Paperback Software) and Exsys (Exsys Corp.).

3.3.2. General-Purpose Tools of Medium Complexity

These are tools that provide multiple representation schemes, and capabilities such as uncertainty handling and knowledge organization. Typically, they also have reasonable interfaces and application integration capabilities. Tools in this category are applicable in most situations that are beyond the simplistic. However, for certain applications, these tools may fall short; for example, if the application is very large, requires real-time capabilities, or extensive custom coding. Typically, medium complexity is also associated with prices that are middle-of-the-road. Two tools that exemplify this category are Nexpert Object (Neuron Data) and Kappa-PC (Intellicorp).

3.3.3. High End Tools and Special-Purpose Tools

These tools are suitable for those applications that have complex requirements, or those that clearly map into a particular type of problem-solving task. This category includes tools that have extensive development capabilities, or features specifically designed to be used for a particular type of task. Examples of such special-purpose features are interfaces to process control systems, temporal reasoning capabilities, or scheduling capabilities. G2 (Gensym Corp.) is a good example of a tool with some of these features. Examples of task-specific tools are TestBench (Carnegie Group) and Design ++ (Design Power, Inc.). ProKappa (Intellicorp) is an example of a tool with fairly sophisticated features, at the high end of the spectrum of representational capabilities.

3.3.4. Others

In rare cases, there may be a fourth category, ie, implementation in a programming language such as Lisp or Prolog. This approach was popular in the early days of AI, primarily because of the rapid prototyping capabilities of such languages. However, from the viewpoint of good software engineering, this is probably not an advisable direction to take. The tool approach is preferable; it has the advantages of explicit representation, built-in capabilities for documentation, explanation and integration, and if needed, access to a programming language as well.

4. Alternative Technologies

For a number of cognitive or interpretive tasks, there are alternatives to mainstream knowledge-based systems that may be more appropriate, especially if adaptive behavior and learning capability are important to system performance. Two approaches that embody these characteristics are neural networks (nets) and case-based reasoning.

4.1. Neural Nets

Neural nets arose out of an alternative approach to solving the problems raised by AI. Instead of modeling high level reasoning processes, neural networks attempt to produce intelligent behavior using the brain as the architectural model. The motivation behind neural network research is the modeling of cognitive activities such as perception, learning, and data interpretation, which symbolic AI approaches do not address adequately. From a practical standpoint, neural nets attempt to address many limitations of knowledge-based systems, including lack of adaptability (self-modification in response to a changing world); lack of robustness (tolerance for missing, bad, or incomplete information); problems of knowledge acquisition (extracting knowledge from experts); problems of storage capacity (amount of knowledge that can be stuffed into a knowledge base); problems of scalability (performance of large systems in comparison to small ones); and problems of speed, especially with increase in the size of the system.

Since biological systems can reasonably cope with some of these problems, the intuition behind neural nets is that computing systems based on the architecture of the brain can better emulate human cognitive behavior than systems based on symbol manipulation. Unfortunately, the processing characteristics of the brain are as yet incompletely understood. Consequently, computational systems based on brain architecture are highly simplified models of their biological analogues. To make this distinction clear, neural nets are often referred to as artificial neural networks.

Neural nets consist of many individual computing elements that are interconnected (61). Each individual computing element, or node, is computationally simple, yet complex behavior can arise out of the multiplicity of the nodes and their interconnectivity. The physical realization of this computing machinery can either be in hardware form, or in the form of a software emulation run on a conventional computer. Instead of symbolic representations, the knowledge inside a neural net is distributed all over the network in the form of connection strengths between computing elements. There is no knowledge engineering; neural nets automatically acquire the connection strengths when they are trained on representative examples of inputs and outputs. Figure 5 shows the architecture and the elemental computing capability of a typical neural net.

Neural net architectures come in many flavors, differing in the functions used in the nodes, the number of nodes and layers, their connectivity, and most importantly, the training algorithm used to make the net learn the connection strengths. The architecture shown in Figure 5 represents one particular type of neural net known as a backpropagation network. Backpropagation networks are the most popular and well-understood types of neural nets, and they have been used in many different types of applications. Other network architectures include self-organizing feature maps, adaptive resonance theory networks, radial basis function networks, and learning vector quantization networks (62–65).

Neural networks have the following advantages: (*1*) once trained, their response to input data is extremely fast; (*2*) they are tolerant of noisy and incomplete input data; (*3*) they do not require knowledge engineering and can be built directly from example data; (*4*) they do not require either domain models or models of problem solving; and (*5*) they can store large amounts of information implicitly.

In general, neural nets are particularly good at pattern recognition, classification, and data interpretation tasks. This makes them especially suitable for a variety of applications: classifying input signal patterns into one or more predetermined categories, eg, identification of compressor problems based on vibration patterns; feature recognition in image processing; identifying categories in raw data, eg, organizing data into related

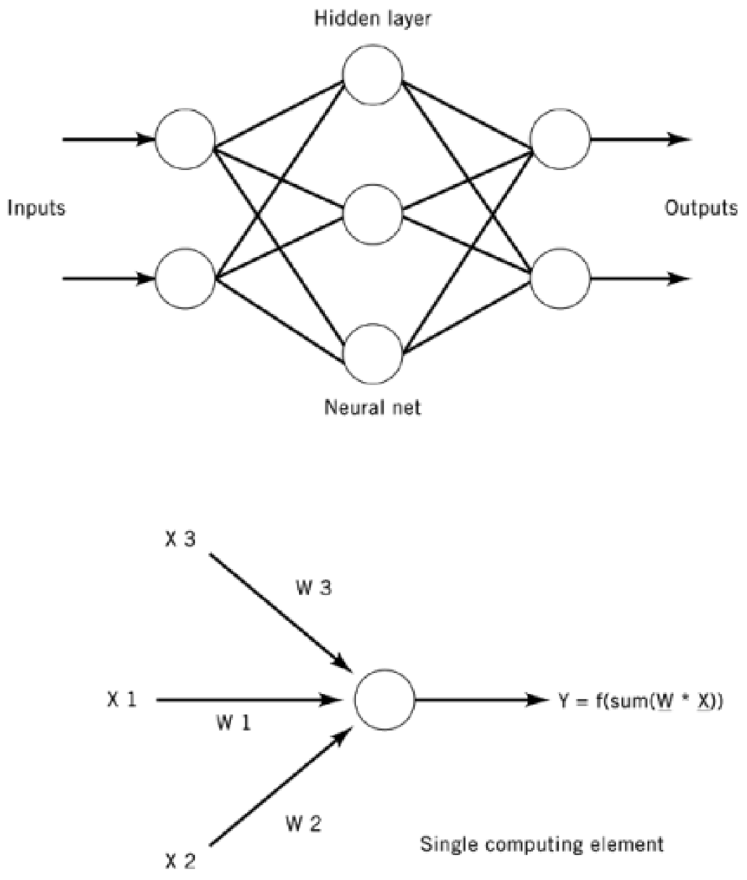

Fig. 5. The backpropagation neural net.

clusters; and complementing knowledge-based systems in areas where KBS technology is notoriously weak, eg, mapping raw data into symbolic abstractions with which the KBS can then reason.

Neural nets can also be used for modeling physical systems whose behavior is poorly understood, as an alternative to nonlinear statistical techniques, eg, to develop empirical relationships between independent and dependent variables using large amounts of raw data.

When considering neural network technology as an alternative to knowledge-based systems, there are certain limitations of neural nets that should be recognized. Neural nets, in general, do not handle time well. Their design does not support end-user interaction, eg, asking the user for missing data. They do not have explanation capability. Their performance is usually highly dependent on the amount and quality of training data used. Some network architectures, eg, backpropagation, require long training times. Many design issues are more art than science, and often have to be resolved by trial and error, eg, the number of nodes and layers to use, the appropriate connectivity between nodes, and the amount of training needed.

In the chemical engineering domain, neural nets have been applied to a variety of problems. Examples include diagnosis (66, 67), process modeling (68, 69), process control (70, 71), and data interpretation (72, 73). Industrial application areas include distillation column operation (74), fluidized-bed combustion (75), petroleum refining (76), and composites manufacture (77).

4.2. Case-Based Reasoning

Case-based reasoningis a relatively new alternative to expert systems. The case-based approach is founded on the observation that humans often solve problems by drawing upon specific cases or episodes from their past experience. As a simplification, one might say that the emphasis in case-based reasoning is on memory rather than knowledge. The types of memory addressed include semantic memory (how static facts, and their relation to one another, are stored), episodic memory (how information about temporal events is stored), and conceptual memory (how concepts are stored in episodic form). By modeling memory, case-based reasoning theory attempts to cohesively explain many aspects of cognition, learning, and problem solving, bridging research in both AI and psychology.

Part of the motivation for case-based reasoning research stemmed from the same limitations of knowledge-based system technology that motivated neural nets. That is, knowledge acquisition from experts can be a notoriously difficult task; knowledge-based systems have neither memory nor evolutionary behavior, ie, they do not remember the results of their problem solving, and they do not learn to perform differently based on past successes or failures; and some knowledge-based systems do not have the ability to degrade gracefully, ie, when confronted with situations that are not explicitly covered in the knowledge base, they fail without providing partial solutions.

However, the way case-based reasoning (CBR) addresses these limitations is fundamentally different from that of neural nets. CBR systems make a fundamental commitment to representing knowledge in the form of episodic cases, which are then retrieved, adapted to new situations, and stored for future reference. The case is the basic unit of knowledge in a CBR system. The knowledge base is the case memory, and the problem solving strategy is the mechanism by which cases are retrieved, modified, applied to the new situation, and then stored back into the case repository. The key computational issues are how to appropriately index cases for retrieval, what methods to use for adapting old cases to new situations, what to do if the adapted case fails to work, and how to store new cases (successful or otherwise) in the case memory. Several good overviews of the technology are available (78, 79). More extensive coverage is provided in recent texts (80).

Case-based reasoning, as a general model of problem solving, has several advantages: knowledge acquisition, although not eliminated, is simplified, eg, only actual episodic cases have to be extracted from the expert, not generalized rules; problem solving behavior can evolve through repeated application of the case memory, ie, CBR systems can learn from experience; and problem solving behavior can be robust, ie, even if retrieved cases fail to apply exactly to a new situation, they may work after modification.

At the same time, there are also many open research issues for which general solutions are not yet available, including the content and structure of cases, the organization and indexing of cases, the problems associated with search as case memories grow larger, the evolution of indexing schemes with experience, metrics for deciding which cases are applicable to a new situation, mechanisms for modifying cases, and mechanisms for repairing solutions when modified cases fail to work. In spite of these open-ended problems, a number of successful applications have been reported (81–84); however, the bulk of these applications have been in academic settings. The technology as a whole has been slow to move into the industrial world. This is partly because of the complexity of the case representation and the unavailability of easy-to-use CBR tools. This contrasts with the representational clarity and programming simplicity that were responsible for the knowledge-based system explosion.

At the present time, there are almost no well-known applications of CBR technology in the chemical engineering domain. A few industrial applications in other domains have been reported (85, 86). The applications cover problem-solving tasks such as scheduling, planning, design, and diagnosis. A few case-based development tools have also emerged. These include CBR-Express (Inference Corp.) and Remind (Trinzic Corp.).

BIBLIOGRAPHY

Cited Publications

1. W. J. Clancey, “Viewing Knowledge Bases as Qualitative Models,” *IEEE Expert* (Summer 1989).
2. E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, Elsevier, New York, 1976.
3. J. McDermott, *Art. Intell.* **19** (1982).
4. R. O. Duda, J. G. Gaschnig, and P. E. Hart, in D. Michie, ed., *Expert Systems in the Microelectronic Age*, Edinburgh University Press, UK, 1979.
5. A. M. Turing, *Mind* **59** (1950).
6. J. Haugeland, *Artificial Intelligence: The Very Idea*, MIT Press, Boston, 1989.
7. C. E. Shannon, *Sci. Am.* **182**, 48 (1950).
8. A. Newell, H. A. Simon, and J. C. Shaw, in E. A. Feigenbaum and J. Feldman, eds., *Computers and Thought*, McGraw-Hill Book Co., Inc., New York, 1963.
9. G. W. Ernst and A. Newell, *GPS: A Case Study in Generality and Problem Solving*, Academic Press, Inc., New York, 1969.
10. R. C. Schank, *Conceptual Information Processing*, North-Holland, Amsterdam, 1975.
11. T. Winograd, *Understanding Natural Language*, Academic Press, Inc., New York, 1972.
12. M. Minsky, *Semantic Information Processing*, MIT Press, Cambridge, Mass., 1968.
13. E. Rich, *Artificial Intelligence*, McGraw-Hill Book Co., Inc., New York, 1983.
14. R. E. Fikes and N. J. Nilsson, *Artif. Intell.* **2** (1971).
15. R. K. Lindsay and co-workers, *Applications of Artificial Intelligence to Chemistry: The DENDRAL Project*, McGraw-Hill Book Co., Inc., New York, 1980.
16. D. L. Chester, D. E. Lamb, and P. Dhurjati, *ASME-Computers in Engineering* **1**, 345–351 (1984).
17. J. Calandranis, G. Stephanopoulous, and S. Nunokawa, *Chem. Eng. Prog.* **86** (1990).
18. T. S. Ramesh, J. F. Davis, and G. M. Schwenzer, *Computers Chem. Eng.* **16**(2), (1992).
19. C. A. Siletti and G. Stephanopoulous, *A Computer Program for Synthesizing Downstream Process Designs*, technical report no. LISPE-88-043, Laboratory for Intelligent Systems in Process Engineering, MIT, Cambridge, Mass., 1988.
20. D. R. Myers, J. F. Davis, and D. J. Herman, *Computers Chem. Eng.* **12**(9/10) (1988).
21. S. H. Rich, A. L. Guerrero, and G. M. Schwenzer, “Knowledge-based Simulation for Operations Analysis and Facilities Design of Refinery Off-Sites,” paper presented at *NPRA Computer Conference*, Seattle, Wash., 1990.
22. T. Weatherill and I. T. Cameron, *Computers Chem. Eng.* **13**(11/12) (1989).
23. V. K. Tzounas and co-workers, *Ind. Eng. Chem. Res.* **29**(3) (1990).
24. R. Lakshmanan and G. Stephanopoulous, *Computers Chem. Eng.* **12**(9/10) (1988).
25. M. Realff, *An Analysis of the Chemical Batch Production Problem and a Detailed Methodology for Scheduling*, technical report no. LISPE-88-054, Laboratory for Intelligent Systems in Process Engineering, MIT, Cambridge, Mass., 1989.
26. M. Hofmeister, L. Halasz, and D. W. T. Rippin, *Computers Chem. Eng.* **13**(11/12), 1255–1261 (1989).
27. J. F. Faccenda and T. E. Baker, “An Integrated AI/OR Approach to Blocked Operations Scheduling,” paper presented at the NPRA Computer Conference, Seattle, Wash., 1990.
28. B. Chandrasekaran, *IEEE Expert* **1**(3), 23–30 (1986).
29. A. Bundy, *The Computer Modeling of Mathematical Reasoning*, Academic Press, Inc., London, 1983.
30. K. Parsaye and K. Y. Lin, “An Expert System Structure for Automatic Fault Tree Generation for Emergency Feedwater Systems for Nuclear Power Plants,” in *Proceedings of the Second IEEE Westex Conference*, Anaheim, Calif., June, 1987.
31. P. Jackson, *Introduction to Expert Systems*, Addison-Wesley, Worthington, UK, 1986.
32. L. A. Zadeh, *Inf. Control* **8**, 338–353 (1965).
33. D. McDermott, *Cognitive Sci.* **6**, 101–155 (1982).
34. D. McDermott and J. Doyle, *Artif. Intell.* **13**(1) (Nov. 1980).
35. T. C. Bylander and S. Mittal, *AI Magazine* **7**, 66–77 (1986).
36. K. Parsaye and M. Chignell, *Expert Systems for Experts*, John Wiley & Sons, Inc., New York, 1988.
37. W. J. Clancey, *Artif. Intell.* **20** (1983).

38. M. C. Tanner and J. Josephson, “Explanation and Abductive Justification,” in *Proceedings of the Spring Symposium Series on Artificial Intelligence in Medicine*, Stanford University, Calif., Mar. 1988.
39. W. J. Clancey in J. F. Traub, ed., *Annual Reviews*, Palo Alto, Calif., 1986.
40. B. Chandrasekaran and S. Mittal, *Intl. J. of Man-Machine Studies* **19**, 425–436 (1983).
41. T. Bylander, “Complexity of Model-Based Reasoning” in T. Bylander, “Complexity of Model-Based Reasoning” *Proceedings of the 1989 AAAI Workshop on Model-Based Reasoning*, Seattle, Wash., 1989.
42. J. McDowell and J. F. Davis, *AIChE J.* **37**(4) (Apr. 1991).
43. V. Venkatasubramanian and S. H. Rich, *Computers Chem. Eng.* **12**(9/10), 903–921 (1988).
44. R. Davis and W. Hamscher, in H. E. Shrobe, ed., *Explorations in Artificial Intelligence*, MIT Press, Cambridge, Mass., 1988.
45. D. Dvorak and B. Kuipers, *IEEE Expert* **6**(3) (June 1991).
46. L. Steels, *AI Magazine* **11**(2) (Summer 1990).
47. R. T. Dodhiawala, N. S. Sridharan, and C. Pickering, in V. Jaganathan, R. T. Dodhiawala and L. Baum, eds., *Blackboard Architecture and Applications*, Academic Press, Inc., New York, 1989.
48. T. J. Laffey and co-workers, *AI Magazine* **9**(1) (Spring 1988).
49. R. Bhatnagar, B. Chandrasekaran, and D. D. Sharma, *Communications of the ACM* **34**(8) (1991).
50. P. A. Sachs, A. M. Patterson, and M. H. M. Turner, *Expert Systems* **3**(1) (1986).
51. S. Padalkar and co-workers, *IEEE Expert* **6**(3) (June 1991).
52. J. E. Clancy, G. J. Carrette, and K. M. Gersten, *ISA Transactions* **31**(3) (1992).
53. S. Weiss and C. Kulikowski, *A Practical Guide to Designing Expert Systems*, Rowman and Allenheld, Totowa, N.J., 1984.
54. E. Payne and R. McArthur, *Developing Expert Systems*, John Wiley & Sons, Inc., New York, 1990.
55. E. Sacerdoti, *AI Expert*, (May 1991).
56. J. R. Slagle and M. R. Wick, *AI Expert*, (Winter 1988).
57. B. Chandrasekaran, *AI Magazine* **11**(4) (Winter 1990).
58. S. K. Shum and co-workers, “A Knowledge-Based System Framework for Diagnosis in Process Plants,” in *Proceedings of the Seventh Power Plant Dynamics, Control and Testing Symposium*, Knoxville, Tenn., 1989.
59. J. S. di Piazza and F. A. Helsabeck, *AI Magazine* **11**(3) (1990).
60. T. A. Nguyen and co-workers, *AI Magazine* **8**(2) (1987).
61. R. Lippmann, *IEEE ASSP Magazine* **4** (Apr. 1987).
62. T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1987.
63. G. A. Carpenter and S. Grossberg, *Appl. Opt.* **26**, 4919–4931 (1987).
64. J. Moody and C. J. Darken, *Neural Computation* **1**, 281–294 (1989).
65. S. C. Ahalt and co-workers, *Neural Networks* **3**, 277–290 (1990).
66. J. C. Hoskins, K. M. Kaliyur, and D. M. Himmelblau, “The Application of Artificial Neural Networks to Fault Diagnosis in Chemical Processing,” paper presented at *AIChE Spring Meeting*, Houston, Tex., 1988.
67. J. J. Ferrada, M. D. Gordon, and I. W. Osborne-Lee, “Application of Neural Networks for Fault Diagnosis in Plant Production,” paper presented at *AIChE National Meeting*, San Francisco, 1989.
68. J. C. Hoskins and D. M. Himmelblau, *Computers Chem. Eng.* **12**, 881–890 (1988).
69. N. Bhat and T. J. McAvoy “Dynamic Process Modeling via Neural Computing,” paper presented at *AIChE National Meeting*, San Francisco, 1989.
70. N. Bhat and T. J. McAvoy, *Computers Chem. Eng.* **14**, 573–583 (1990).
71. L. H. Ungar, B. A. Powell, and S. N. Kamens, *Computers Chem. Eng.* **14**, 561–572 (1990).
72. J. R. Whiteley and J. F. Davis “Qualitative Interpretation of Sensor Patterns using a Similarity-Based Approach,” paper presented at the *IFAC Symposium on On-Line Fault Detection and Supervision in the Chemical Process Industries*, Newark, Del., Apr. 1992.
73. T. J. McAvoy and co-workers, “Interpreting Biosensor Data via Backpropagation,” in *Proceedings of the International Joint Conf. on Neural Networks*, Washington, D.C., 1989.
74. D. Blanchard, *AI Week*, (Sept. 1, 1990).
75. S. Moorthy and P. Iyer, “Neural Network Simulation for Plant Control Optimization,” in *Proceedings of the Industrial Computing Conference*, ISA, Anaheim, Calif., 1991.
76. G. Hobson, “Neural Network Applications at PSP,” paper presented at *NPRA Computer Conference*, Seattle, Wash.,

1990.
77. D. A. Sofge and D. A. White, “Neural Network Based Process Optimization and Control,” in *Proceedings of the 29th Conference on Decision and Control*, 1990.
78. S. Slade, *AI Magazine* **12**(1) (1991).
79. J. L. Kolodner, *AI Magazine* **12**(2) (1991).
80. C. Riesbeck and R. C. Schank, *Inside Case-Based Reasoning*, Lawrence Erlbaum, Hillsdale, N.J., 1989.
81. E. Bareiss, B. Porter, and C. Wier, *J. of Man-Machine Studies* **29**, 549–561 (1988).
82. K. Sycara, “Using Case-Based Reasoning for Plan Adaptation and Repair,” in *Proceedings of the DARPA Workshop on Case-Based Reasoning*, Morgan Kaufman, San Mateo, Calif., 1988.
83. D. Navinchandra, “Case-Based Reasoning in CYCLOPS, A Design Problem Solver,” in Ref. 82.
84. R. Turner, “Organizing and Using Schematic Knowledge for Medical Diagnosis,” in Ref. 82.
85. P. Koton, “SMARTPLAN: A Case-Based Resource Allocation and Scheduling System,” in *Proceedings of a Workshop on Case-Based Reasoning*, Morgan Kaufman, San Mateo, Calif., 1989.
86. W. Mark, “Case-Based Reasoning for Autoclave Management,” in Ref. 85.

General References

87. Artificial intelligenceJ. Haugeland, *Artificial Intelligence: The Very Idea*, MIT Press, Boston, Mass., 1989.
88. E. Rich, *Artificial Intelligence*, McGraw-Hill Book Co., Inc., New York, 1983.
89. P. H. Winston, *Artificial Intelligence*, Addison-Wesley, Reading, Mass., 1984.

Expert systems

90. J. P. Ignizio, *Introduction to Expert Systems*, McGraw-Hill Book Co., Inc., New York, 1991.
91. P. Jackson, *Introduction to Expert Systems*, Addison-Wesley, Worthington, UK, 1986.
92. K. Parsaye and M. Chignell, *Expert Systems for Experts*, John Wiley & Sons, Inc., New York, 1988.
93. M. Van Horn, (The Waite Group), *Understanding Expert Systems*, Bantam Books, New York, 1986.
94. D. A. Waterman, *A Guide to Expert Systems*, Addison-Wesley, Reading, Mass., 1986.

Expert system development

95. P. Harmon and R. Maus, *Expert Systems Tools and Applications*, John Wiley & Sons, Inc., New York, 1989.
96. E. Payne and R. McArthur, *Developing Expert Systems*, John Wiley & Sons, Inc., New York, 1990.
97. S. Weiss and C. Kulikowski, *A Practical Guide to Designing Expert Systems*, Rowman and Allenheld, Totowa, N.J., 1984.

Expert systems in chemical engineering

98. R. H. Cadmus and M. D. Woolsey, *Oil & Gas J.*, (Jan. 9, 1989).
99. *Chem. Eng. Prog., Expert Systems Issue*, **83**(9) (1987).
100. M. Mavrovouniotis, ed., *Artificial Intelligence Applications in Process Engineering*, Academic Press, Inc., New York, 1990.

101. D. A. Rowan, *AI Expert*, (Aug. 1989).
102. G. Stephanopoulous, *Computers Chem. Eng.* **14**(11), (Nov. 1990).

T. S. RAMESH
Mobil Research and Development Corporation

Related Articles

Computer-aided design; Computer technology