Engineering with Computers 1, 9–25 (1985)

Engineering with Computers

© Springer-Verlag 1985

EXPERT SYSTEMS: New Approaches to Computer-aided Engineering*

Clive L. Dym
Knowledge Systems Area, Intelligent Systems Laboratory, Xerox PARC and Department of Civil Engineering, University of Massachusetts

Abstract. This paper provides an overview of the burgeoning new field of expert (knowledge-based) systems. This survey is tutorial in nature, intended to convey the *gestalt* of such systems to engineers who are newly exposed to the field. The discussion includes definitions, basic concepts, expert system architecture, descriptions of some of the programming tools and environments with which knowledge-based systems can be built, and approaches to knowledge acquisition. Some currently extant expert systems are described *en passant*, including a few developed for engineering purposes. Comments follow on the engineering of knowledge, as both cultural and social processes. The paper closes with an assessment of the roles that expert systems can play in engineering analysis, design, planning, and education.

1 Introduction

This paper is intended to provide an overview of what expert systems are, what they are not, what their characteristics and features are, and something about how they are built. This necessitates discussing what knowledge is, how it can be shared, and how it can be engineered, for the hallmark of an expert system is that it is an attempt to capture in a computer program expertise and knowledge as displayed by human experts. This also requires that different programming languages and paradigms be discussed, as well as different program architectures and programming environments.

There are hardware implications as well, which are touched on briefly, especially as they are related to the Japanese Fifth Generation Project [1]. Beyond expert systems there is new technology, hardware and software, which also has tremendous implications and potential for the sharing of knowledge, for integrated design and manufacturing, and so on, and a few words are said about this. And, finally, the paper concludes with suggestions of the opportunities for engineering practice and education that are raised by the development of expert systems and concurrent technology.

The starting point for this discussion is a pair of quotations. The first is from a book [1] whose senior author is often identified as the father of the field of expert systems, E.A. Feigenbaum:

> Even though a lot of professional work seems to be expressed in mathematical formulas . . . the matters that set experts apart from beginners are symbolic, inferential, and rooted in experiential knowledge. Human experts have acquired their expertise not only from explicit knowledge found in textbooks and lectures, but also from experience: by doing things again and again, failing, succeeding . . . getting a feel for a problem, learning when to go by the book and when to break the rules. They therefore build up a repertory of working rules of thumb, or "heuristics," that, combined with book knowledge, make them expert practitioners.

The second quotation derives from a paper by Stefik and Conway [2] that describes part of an experiment in knowledge engineering, about which more will be said later:

> Knowledge is an artifact, worthy of design.

And so we begin.

2 What Is an Expert System? What Does It Do?

The roots of expert systems lie in the field of artificial intelligence (AI), which may be said to be the science that tries to create intelligent behavior on computers. In studying knowledge and its use, we find many interesting features [3]. For example, it turns out that expertness generally comes in narrow, highly specialized fields. Further, what distinguishes an expert is the rapidity with which he or she recognizes patterns and brings appropriate rules to bear. Although these rules are often heuristic, it is their use to narrow the search of a solution space that is important. But, most convincing about expertise is the ability of the expert to

* The paper was presented as the (invited) SDM Lecture at the Twenty-Fifth AIAA-ASME-ASCE-AHS Structures, Structural Dynamics and Materials Conference in Palm Springs, CA, May 15, 1984, and appeared in the proceedings of that meeting.

Reprint request: Clive L. Dym, Department of Civil Engineering, University of Massachusetts, Amherst, MA 01003.

explain the reasoning path by which he arrived at the solution to a problem.

Specialists in AI also divide knowledge into "surface" and "deep" components [4]. The term *deep knowledge* is used to refer to reasoning from basic principles, that is, from basic laws of nature or structural and behavioral models. *Surface knowledge* is that heuristic, experiential knowledge that comes from having successfully solved a lot of problems. In current usage, although surface knowledge is often very useful, it is regarded as being based on experience, rather than being grounded in first principles or on deep knowledge. (The distillation of deep knowledge into an optimized, efficient form results in *compiled knowledge*, which can be confused with surface knowledge. The ability to explain heuristics in terms of first principles, for example, indicates compiled knowledge, whereas the application of the same rules simply because they work is typical of surface knowledge.) In our own work in engineering, we might recognize here a very loose analogy between having access to a full-blown finite element package, on the one hand, that might solve any well-posed problem in solid mechanics, and having available, on the other hand, an experienced consultant who seems to be able to identify and solve the critical problems on the back of an envelope. Even if unfortunately and wrongly characterized as surface knowledge, it is that compiled expertise that is most useful for knowing when and how to apply deep knowledge and is therefore most prized. That expertise is also prized because it can often be applied beyond the limits of existing formal theory.

An *expert system* is a computer program that performs a task normally done by an expert or a consultant, and in so doing it uses captured, heuristic knowledge. If we think of a computer program as a sequence of rules for action, then the familiar algorithmic program may be viewed as one where the sequence of *firing*, or testing and applying, the rules is determined in advance and where each rule premise leads to one, and only one, action. That is, the progress of such a program is controlled by a tightly knit algorithm. Further, in a conventional algorithmic program, the processing is done by using symbols to represent numbers, arithmetic properties, and mathematical operations.

In an expert system, by way of contrast, the sequence of firing of rules is determined within the program by an inference engine, and the rule premises may lead to multiple actions or no action at all. Both the inference engine and the rules may incorporate heuristics, rules of thumb, that are accumulated by an expert after years of problem solving. This allows an expert system to reason as it performs a task, as well as adapt to new situations. The symbols in an expert system are used to represent virtually any kind of object, including people, materials, biological organisms, classes of objects, concepts, and so on. We will see specific examples later on, but the interesting point here is that the desire to represent and manipulate symbols as other than numbers requires different programming language features and facilities.

An excellent concise definition of an expert system has been given by Brachman et al. [3]:

> An expert system is one that has expert rules and avoids blind search, reasons by manipulating symbols, grasps fundamental domain principles, and has complete weaker reasoning methods to fall back on when expert rules fail and to use in producing explanations. It deals with difficult problems in a complex domain, can take a problem description in lay terms and convert it to an internal representation appropriate for processing with its expert rules, and it can reason about its own knowledge (or lack thereof), especially to reconstruct inference paths rationally for explanation and self-justification.

What does an expert system do? A better question is: What sort of tasks can expert systems do? Knowledge-based systems have been built to *interpret, diagnose, monitor, predict, instruct, plan,* and *design.* Examples of particular systems follow, but it is important to recognize that these tasks are very different in their respective formulations, and thus will require different architectures for searching their solution spaces [5]. For example, for interpretation and diagnosis, the requisite bits of knowledge about the solution that is sought are in the expert system's *knowledge base.* What the system does is construct a reasoned path to the actual solution, that is, the system *derives* the solution. For planning and design, the knowledge base contains the properties of the desired solution or the constraints within which a solution must be found. Here, however, the system must generate and test solutions; that is, the system *forms* solutions.

3 How Does an Expert System Work?

In this section, the first of three devoted to the architecture of expert systems, we describe the basic structure of an expert system, following which we discuss some of the elements of the internal architecture that define how the components do their own tasks and how they interact. We will refer here to a particular expert system to illustrate one possible architecture for a knowledge-based system, as well as provide a context for pointing out alternative organizations. The particular system is called SACON, which is an expert system that consults with the user on the appropriate

use of the MARC finite element code for structural analysis [6,7]. The next sections deal with expert system architecture in more general terms, wherein we relate architectural issues to the types of tasks the system is meant to perform.

The basic structure of an expert system appears in Fig. 1 (after Feigenbaum and McCorduck, p. 76 [1]). We discuss the components first and the roles of the interacting people second. The components include: *input/output facilities,* which allow the user to communicate with the system and to create and use a data base for the specific case at hand; an *inference engine,* which incorporates reasoning methods, which in turn act upon input data and knowledge from the knowledge base, to both solve the stated problem and produce an explanation for the solution; a *knowledge base,* which contains the basic knowledge of the domain, including facts, beliefs, and that heuristic lore unique to the expert; and (perhaps) a *knowledge acquisition facility,* which allows the system to acquire further knowledge about the domain from experts, or even automatically, from libraries, data bases, and so on. The inference engine acts as the executive that runs the expert system. It fires rules according to a built-in reasoning protocol, and by so doing performs actions that lead to solution of the problem and, at the same time, may change the knowledge base by adding new knowledge to it. The knowledge base contains the domain-specific knowledge, which provides the context for the specific applications of the expert system. The inference engine and the knowledge base are not always maintained as entirely separate components, and we will see that there is a lot of overlap and interaction between the concepts underlying both.

Separating the domain-specific knowledge from the inference engine provides an interesting and potentially powerful incentive to builders of expert systems; that is, the same inference engine can be used together with knowledge bases drawn from different domains to make different expert systems from the same basic building blocks. This idea has already been advantageously exploited (see, e.g. Bennett and Englemore [6] and Bennett et al. [7], where the skeletal system EMYCIN was used to develop a structural analysis consultant system called SACON). It has also led to research in and development of system building environments, or kits, which can be used in different domains to build systems that do different kinds of tasks. We will return to this matter later.

Note that three basic players are depicted in Fig. 1: the user of the system, the domain expert, and the knowledge engineer. The *knowledge engineer* is the system builder, that is, the expert in AI techniques who structures the expert's knowledge so that it can

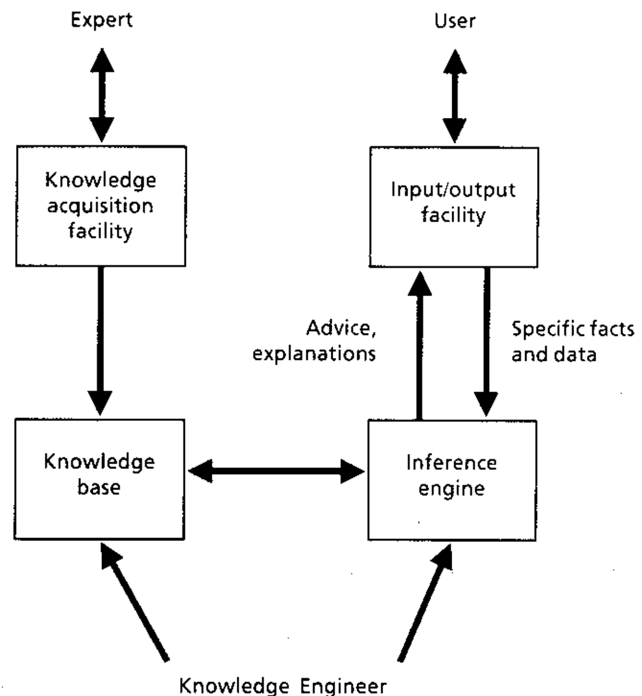

Fig. 1. Basic structure of an expert system (After Feigenbaum and McCorduck, p. 76 [1])

be shared by the user. There are some very interesting issues regarding the creation, structuring, and sharing of knowledge, some of which can be phrased in terms of questions about who the players are and how they interact. One kind of question is whether the user need be as expert as the expert who supplied the knowledge. That is, can systems be created for use by individuals less experienced or expert than the domain expert? The answer to this question is, "Yes," and there are examples in medicine [8] and engineering [6] of systems designed to be used by practitioners of related but not identical expertise in the first case, and by relative novices in the other.

Another question is whether the domain expert can also be the knowledge engineer? Or, can a domain expert build his own expert system? The answer to this question is not entirely without controversy, as some AI researchers imply that the process would not work, that an expert would not be able to successfully articulate his knowledge on his own [9], whereas others claim that there is no a priori, logical reason that would prevent a domain expert from being his own knowledge engineer [10]. There are two pragmatic dangers worth noting, however. One is that domain experts wishing to build their own system must learn a lot about knowledge representation, as will become clearer subsequently, and they should not underestimate the enormity of the task. On the other hand,

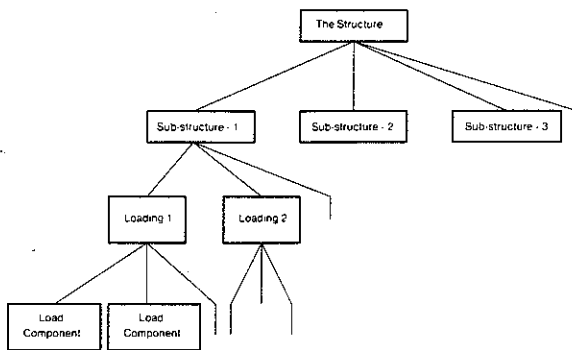

Fig. 2. Context tree for SACON (After Bennett et al., Fig. 2.1 [7])

although knowledge engineers learn a lot about the domain while building a system, they remain at best talented amateurs in that field and thus should remember that they are not domain experts.

To discuss the makeup of the knowledge base and inference engine in greater detail, we must focus on the nature of the problems being solved, or the tasks undertaken. And, again, we will use SACON as our base example for this discussion. We need a way to express both the intelligent task we wish the expert system to do and the knowledge with which the system will exhibit its captured expertise. There are three central, enabling concepts: symbols and symbol structures; a calculus for manipulating symbol structures; and search techniques for successfully reducing the combinatorics in searches of solution spaces, especially very large solution spaces. It is sufficient to think of *symbols* as strings of characters and of a symbol structure as a kind of data structure called a *list structure,* which contains symbols [5]. Thus, a symbol structure might be:

($AND (SAME CNTXT MATERIAL (LIST OF METALS)))

This structure, from the expert system SACON [7], is part of the antecedent (premise) of a rule and says: "And in this context the material of the structure is one of those on the list of metals . . . ." A major advance in AI research was the development of list processing languages, such as *Lisp,* which had operators for manipulating lists and facilities for storing (and retrieving) them [11].

The *predicate calculus* is a formal language of symbol structures that can be used for computer representation of knowledge and of inferential reasoning. One of the adaptations of predicate calculus that is used for inferential reasoning is the construction of rules of the form *IF-THEN* (or *premise-action,* or *antecedent-consequent*) in which the satisfaction of one or more premises leads to one or more actions or consequences. For example, in the expert system SACON, each premise (or clause of a premise) has the following structure [7]:

⟨predicate function⟩ ⟨object⟩ ⟨attribute⟩ ⟨value⟩

In the example of symbol structure cited earlier, the predicate function is SAME, which is one of 24 such domain-independent predicate functions in the program, some of the others being KNOWN, DEFINITE, BETWEEN*, and so on. Each predicate function has as its argument the *associative triple,* "⟨object⟩ ⟨attribute⟩ ⟨value⟩," which assigns a value to a particular attribute of an object. The associative triple is domain-specific in that the objects (e.g., STRUCTURE, LOADING), attributes (e.g., GEOMETRY, MATERIAL), and associated values (e.g., CURVED, ALUMINUM) must be derived from the domain in which the system is operating. The symbol CNTXT in the preceding rule defines which object is referred to by reference to a context tree (see Fig. 2). For SACON, as we discuss in more detail later, the domain is structural mechanics. As another illustration of the use of symbol and list manipulation, we note that in SACON each fact about the world is represented as a 4-tuple list consisting of an associative triple and its current certainty factor, for example:

(SS-STRESS SUB-STRUCTURE-1 FATIGUE 1.0)

This list is a statement that it is known with certainty factor 1.0 that fatigue dominates the state of stress in the part of the structure identified as SUB-STRUCTURE-1.

The inference process begins to appear when the complete rules, called *production rules,* are assembled for the system's knowledge base. For example, Rule 064 of SACON is:

```
PREMISE: ($AND (BETWEEN* CNTXT ERROR 5 30)
               (GREATERP* CNTXT ND-STRESS .7))
ACTION: (CONCLUDE CNTXT SS-NONLINEARITY
         MATERIAL 1.0)
```

This rule conjunctively associates (with the symbol $AND) two premises about a substructure; that is, the tolerable analysis error in percent is between 5 and 30 and the nondimensional stress in the substructure is greater than 0.7. If both are true, the inference to be drawn is the conclusion that material nonlinearity is one of the types of nonlinearity in the substructure, with a certainty factor of 1.0.

The organization of the knowledge base in this instance is completed by representing the objects in hierarchical form in a context tree (Fig. 2). This tree represents the relationships between objects and unfolds dynamically as SACON is used in a consultation. It is called a context tree because it establishes for each rule what is specifically meant by the symbol

CNTXT as the rule is invoked. That is, in the language Lisp in which SACON is written, CNTXT appears as a free variable in the rule structure and is *bound* (assigned) to a particular value depending on the context—from the tree—in which the rule is fired [6].

Now we come to the establishment of a procedure for firing rules, that is, the inference engine. There are several ways to organize this process, among them is the strategy of backward-chaining used in SACON. A *backward-chaining* strategy requires selecting a goal and then scanning the rules to find those whose consequent actions will achieve that goal. For example, if the program is trying to establish the nonlinearity characteristic of a substructure, it will retrieve all rules that include SS-NONLINEARITY in their action (or consequent), and then will invoke each rule in turn, evaluating the premises of each rule to see if all the antecedent conditions have been met. In the sample ruled cited earlier, this means that the system will set up two subgoals in turn—determination of the allowable analysis error (ERROR) and determination of the value of the dimensionless stress (ND-STRESS)—and the process of trying to satisfy a goal recurs. This means that the program is doing a depth-first search of a goal tree. The SACON program also uses the certainty factors and some partial evaluation of rule premises to make the search of the goal tree more efficient, but we will not delve into that level of detail here. We note as well that the strategy of backward-chaining is also called *goal-driven*, and sometimes is referred to as *consequent reasoning*.

4 How Is an Inference Engine Organized?

It is clear from the preceding discussion that SACON (and its predecessors MYCIN [8] and EMYCIN [12]) is based on an inference engine that is goal-driven, and it reasons backward from goals to the data specific to the case. An alternative strategy is *forward-chaining*, which would use a rule base to reason from the input data forward to the conclusions. Also known as *data-driven*, this protocol can be viewed as antecedent reasoning. It has been successfully applied in an expert system called R1, also known as XCON, which was developed for the Digital Equipment Corporation to assist in the configuration of VAX computers [13]. The program determines the physical layout of and the connections between the components for every VAX that is sold, each of which is custom-tailored to the customer's requirements.

What about different inference engines? Perhaps even more than the choice of a knowledge representation scheme, the choice of an inference engine is strongly coupled to the nature of the task the system is

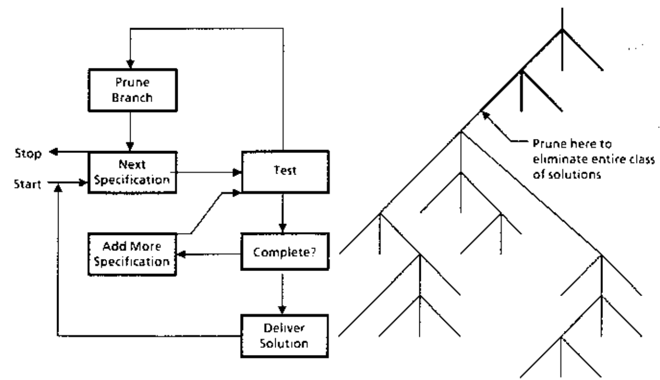

Fig. 3. Hierarchical pruning of classes of solutions in generate-and-text search (After Stefik et al., Fig. 3.3 [5])

designed to perform. That is, since the completion of a task requires the search of a solution space, the search will depend on the size and character of the solution space, tentative reasoning about the domain, and whether or not the data are noisy or varying over time [5]. For example, in *design* tasks, the solution spaces are generally very large, and tentative reasoning is very important because simplifying assumptions (abstractions) are frequently invoked to reduce the magnitude of the problem. Some of the initial assumptions are bound to fail as a design evolves, so it is worthwhile to employ methods that permit the reformulation of assumptions (and their consequences) as further design iterations are completed. The search technique for a design problem may be of the *generate-and-test* variety, for example, in which the system generates possible solutions and then tests them against posted constraints or requirements, pruning those that fail the specified tests. The generation of solutions might use forward-chaining as an inference mechanism, proceeding forward from given data.

Clearly, such a search technique could easily get out of hand, and so effective pruning techniques need to be applied together with a generate-and-test approach. The pruning technique might use backward-chaining, inferring from partial goals to see whether they are satisfied. One such pruning technique, illustrated in Fig. 3, is an hierarchical scheme that prunes on the basis of partial design descriptions, thus permitting the pruning of entire classes of solutions [5]. On the other hand, in a design problem, as in a planning problem, it is often difficult to prune very early because there may not be a reliable way to evaluate partial solutions that may be expressed in fragmentary form [14].

One very important point needs to be mentioned here, in the context of reasoning strategies. We noted earlier that a central characteristic of an acknowledged expert is his or her ability to explain how he or she reached a conclusion. This is equally true of expert

Table 1. A rough characterization of some engineering tasks and corresponding search prescriptions (After [14])

	Analysis	Synthesis
	Interpretation, Diagnosis Monitoring	*Planning, Design*
Task requirements	Unreliable data Time-varying data	Evaluation of partial solutions Fixed sequencing not possible Subgoals interact Need guessing, multiple sources
Search prescriptions	Evidence from multiple sources Probabilistic models Fuzzy models Exact models State-triggered expectations	Generate and test Search space abstractions Constraint propagation Least commitment Plausible reasoning Dependency-directed backtracking

systems: they must exhibit the quality of *transparency*. That is, the user of an expert system should be able to see the chain of reasoning that led to a given outcome. (Computer scientists talk of leaving an *audit trail* so that the user can see the reasoning path.) Indeed, one may even want to adjust the explanation of the reasoning to fit the needs of a particular class of users. This means more than simply knowing whether the system is forward-chaining, or goal-driven, or whatever. It means that the system should be able to display the particular facts, beliefs, inferences, heuristics, rules, and so on that were used, and in what sequence, in order to arrive at a particular conclusion. For example, the ability of the medical diagnostic system MYCIN to explain its behavior was a major reason that this program was acceptable to physicians, in contrast to statistically grounded diagnostic aids [15]. The quality of transparency will surface again in a subsequent discussion of programming environments.

It is impossible to be exhaustive in a discussion such as this. Obviously, the performance of an expert system in executing a particular task will depend heavily on the size of the search space, *ab initio* reasoning, and the nature of the data. So too the search technique, the inference engine, must be strongly coupled to the type of task undertaken. In Table 1 we have tried to characterize the different tasks that expert systems might perform in terms of the characteristics just outlined. A more comprehensive discussion is given by Stefik et al. [5,14].

5 How Is a Knowledge Base Organized?

In both SACON and R1 the knowledge base is encapsulated in a set of rules and an auxiliary data base. Each *production rule* represents a single "chunk" of knowledge about a field, expressed in antecedent-consequent form, and a typical knowledge base contains hundreds of rules. SACON, for example, contains 170 rules and 140 consultation parameters (e.g., loadings, load components, materials, etc.), whereas R1 contains some 800 rules and information about the properties of some 400 VAX components. Thus, for the diagnostic task of SACON as well as the design task of R1, an adequate scheme of knowledge representation is a rule base. However, the inference engine in each is different, with the diagnostic task requiring a goal-driven search and the design task requiring a data-driven search.

Both SACON and R1 use production rules, but there are other, different schemes of *knowledge representation*. Other techniques for representing knowledge include [5,14,16,17]: *semantic networks,* which consist of a set of *nodes* that represent objects, concepts, events, and so on, and of *links* that connect the nodes and represent their interrelations; and *frames,* which are data structures that include declarative and procedural information about an object in predefined *slots* and are normally embedded in frame systems. Examples of a semantic network and of a frame are displayed in Fig. 4.

Semantic networks typically incorporate the mechanism of *inheritance lattices,* which allow for the inheritance of information about objects farther up (more abstract) in the lattice. A *frame system* combines the network and inheritance features of a semantic net with the representation of objects, and so on by frames instead of (atomic) symbols. Frame systems thus allow for the inheritance of information about objects from classes of objects that are farther up in the lattice. The inherited information can include properties, applicable methods, and invocations of specific instances. They also can include *default* values for properties that state expectations about objects, for

example; commercial aircraft might have "jet engines" in the slot for the source of power, even though propeller-powered aircraft still exist. We will display an inheritance lattice a bit later (Fig. 6) when we discuss the programming environment LOOPS, which incorporates an extended concept of a frame system as a representational tool [18,19].

There are advantages to each of these knowledge representation ideas, and thus it has often been worthwhile to combine several such schemes into a single expert system. Production rules, for example, simplify the generation of explanations and of prompts to the user because it is easy to turn an antecedent-consequent (IF-THEN) rule into a question [20]. A rule that states "IF the nondimensional stress is greater than 0.7 THEN the nonlinearity is a material nonlinearity" can be recast as an explanation ("A nondimensional stress greater than 0.7 indicates a material nonlinearity.") and as a question ("Is the nondimensional stress greater than 0.7?"). Thus, an expert system might use a semantic network to relate objects to each other, a frame system to expose the attributes of individual objects, and a set of production rules to uncover attributes or properties of objects. We will discuss such integration schemes in the next section, along with other knowledge engineering tools.

In fact, the choice of a knowledge representation scheme is likely to be the first order of business in constructing an expert system. This is simply because a good start for the knowledge engineer is to learn both the vocabulary of the specific domain, as well as something of those things about which the expert system will reason [21]. The process of structuring his new

Fig. 4(a). Semantic nets showing that B747s have jet engines (After Barr and Davidson, p. 148 [16])

	Generic AIRPLANE Frame Values	B747 Frame Values
Self:	a COMMERCIAL PLANE	an AIRPLANE
Owner:	an AIRLINE (default = American)	TWA
Contents:	CARGO (default = PASSENGERS)	PASSENGERS
Name:	an AIRPLANE TYPE (default = B707)	B747
Manufacturer:	an AIRFRAME MANUFACTURER	Boeing

Fig. 4(b). Generic and specific frames for aircraft (After Barr and Davidson, p. 159 [16])

knowledge in his own mind enables the knowledge engineer to think about its representation in the system. Consistent with the notion of a frame system, for example, one could define things in terms of objects and attach to these objects various instances (exemplars), methods, variables, and inheritance tracers. We show in Fig. 5 the structure of an object from the LOOPS environment mentioned earlier [18]. This is a sample object, actually a class of objects called "AreaBudget." This organization shows that the class

```
[DEFCLASS AreaBudget
    (MetaClass Class      EditedBy (* dgb "15-Feb-82 14:32 ")
                  doc
(* * This is a sample class chosen to illustrate the syntax of classes in
LOOPS. Commentary on the class is inserted in a standard property in the
class. -- e.g. Budgets are . . .))
    (Supers OwnedObject Budget)
    (Class Variables (maxBase 25000))
    (InstanceVariables
       (owner #$VLSI doc (* organizational area that owns budget))
       (base 1000 doc (* The initial amount of money))
       (overhead 2.25 doc (* Multiplied by base to get total.))
       (employees NIL doc (* list of employees in this area))
       (manager NIL doc (* manager of this area))
       (total #(SHARED getTotal UpdateNotAllowed)
          doc (* value of total is computed using active value.))
    (Methods
(Report AreaBudget.Report doc (* Prints out a budget report))
(StoreBase AreaBudget.StoreBase
        doc (* store base value checking maxBase))]
```

Fig. 5. Definition of a class in the LOOPS representation

has two super classes from which it inherits variables and methods and that it has class and instance variables, and methods, which are defined for this class. This structure hints at the power of extended frame systems and shows how the organization of basic information about the domain is reflected in the implementation of a knowledge representation scheme.

6 How Do You Build an Expert System?

There are (at least) two answers to the question phrased in this title. One is an articulation of the issues encompassing the knowledge acquisition process, especially in terms of interactions between domain expert and knowledge engineer, institutional constraints, and so on. These issues are discussed in the next section. A second response would speak to the availability of knowledge engineering tools, languages, facilities, programming environments, and the like, and we examine these aspects now.

Knowledge engineering tools, that is, the tools for capturing knowledge and building expert systems, can be said to incorporate three elements: the programming language; skeleton expert systems, which can be adapted to different domains of expertise; and programming environments, which are software environments created to facilitate program development and prototyping. The domain expert and the system user will see reflections of the choice of tools in the facilities available in the interface to the expert system, that is, in the language used, the facilities for explanation and for auditing of reasoning chains, ease of use, graphics, and so on. We will outline some of the choices available in general-purpose programming languages, skeletal systems, and programming environments, which in this context are sometimes referred to as knowledge representation languages [22]. Greater detail and specific applications, including a side-by-side comparison of several tools, have been given by Barstow et al. [22] and Waterman and Hayes-Roth [23].

One way to start would be to build an expert system from the ground up, using none of the available tools, choosing only a general-purpose programming language to start. More likely than not, the initial language choice would be Lisp, the list processing language discussed earlier, because of its power for symbolic representation and manipulation. Also, many highly interactive environments have been developed on top of Lisp, and these greatly enhance program development and use. Lisp comes in several *dialects*, with the most widely used being various descendants of *Interlisp* [24] and *Maclisp* [25]. There is no clear-cut choice between these two languages, as noted by Barstow et al. [22]:

> The choice of one of them (or of any other Lisp system) is probably more a matter of personal preference and technical availability than of clear technical superiority, although discussions among advocates of Interlisp and Maclisp often take the tone of theological disputes.

It is also worth noting that the dispute, as it were, is over more than just language. Each of the languages has built on it a programming environment with different features and facilities, some of which are inherent to the languages and some of which represent choices by the developers of each environment [26].

There are also choices beyond the various dialects of Lisp. One that is still within the category of symbolic languages is Prolog [27], a logic-oriented language based on first-order predicate calculus. Although not in widespread use in this country, it is the language chosen by the Japanese for their Fifth Generation Project [1]. Finally, one could resort to general-purpose languages such as Fortran and Pascal, which offer the advantages of easy portability among machines as well as compatibility with other types of software, for example, finite element packages. Indeed, even among researchers in AI it is now possible to contemplate—although not seriously advocate—writing expert system programs in general-purpose languages other than the principal symbolic languages of AI [28].

In our discussion of SACON, we noted that it was based on the skeleton of a system, EMYCIN, which was originally the backbone of the medical diagnosis program MYCIN. There are other such skeletons available, including KAS, derived from the geological exploration system PROSPECTOR, and EXPERT, derived from CASNET/GLAUCOMA, an expert system used to diagnose and recommend therapy for glaucoma. In such a skeletal system, the domain knowledge is incorporated into the production rules (all of the preceding systems are rule-based), and the search strategy embodied in the inference engine remains domain-independent; that is, it stays the same regardless of the domain in which the system is to be applied. Therefore, in adapting a (rule-based) skeletal system to a new task, one must make sure that production rules are still suitable for knowledge representation in the new domain and that the inference engine is appropriate to the new task. Each of the aforementioned skeletal systems has its own characteristics (e.g., EMYCIN has a well-engineered environment for developing the knowledge base and good explanation features, KAS allows both forward- and backward-chaining and also has a very good environment for

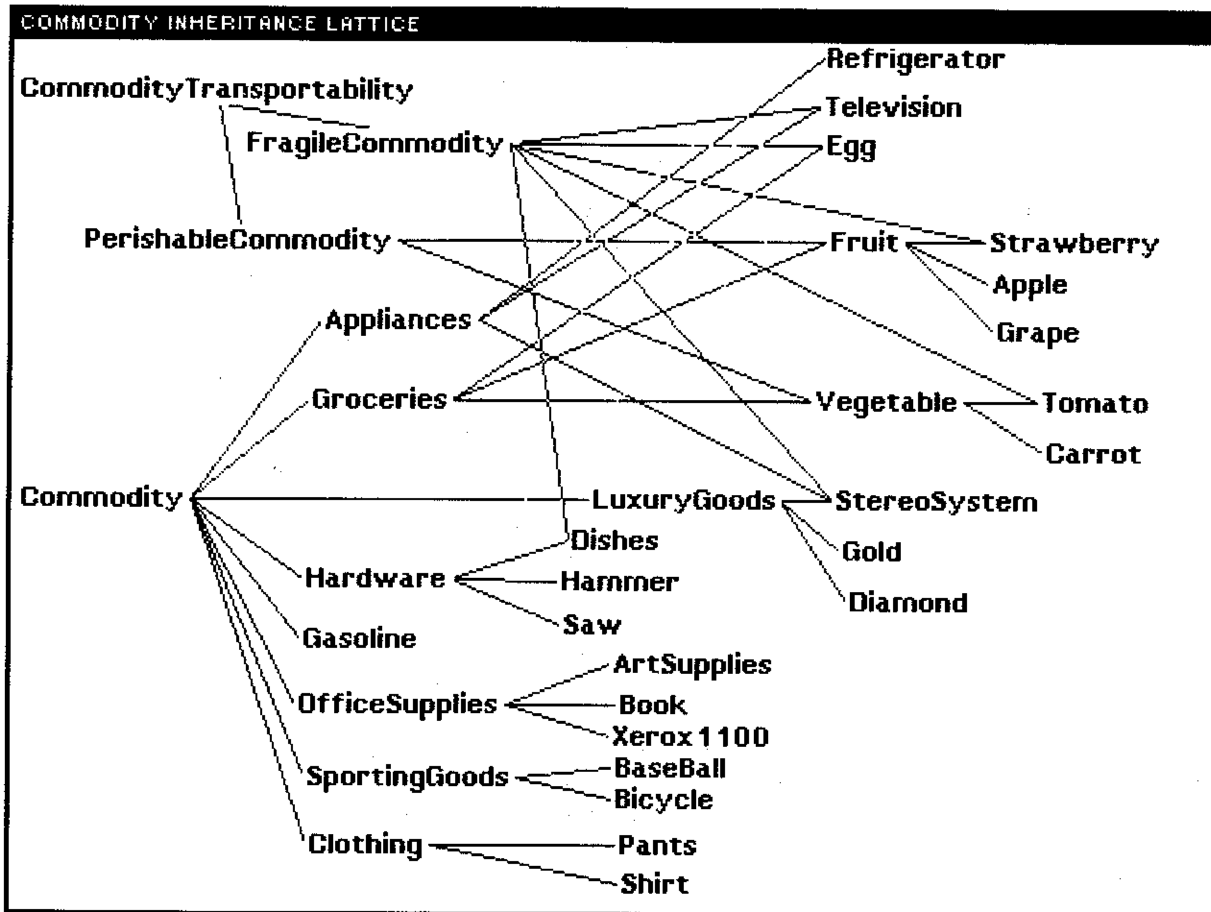

Fig. 6. An inheritance lattice from LOOPS (After Stefik et al., Fig. 9 [19])

debugging rules, and so on, and EXPERT has been programmed in Fortran and is extremely efficient), and the user must thus select a skeletal system carefully, with due attention to both the structure of the task to be performed and the facilities desired in the expert system [22].

Finally, in this section, we note that there are available several general-purpose representation languages, such as ROSIE, OPS5, and HEARSAY-III. Each of these was developed specifically for knowledge engineering applications. Further, these systems are not so closely tied to particular programming paradigms as are the preceding skeletal systems; that is, they allow a greater variety of search structures. Thus, these programming environments are probably less constrained than the skeletal systems, and individually are perhaps each more easily applicable to a greater range of expert system tasks. On the other hand, this lack of constraint may make them harder to apply to a specific task than an obviously appropriate skeletal system. An overview of five such knowledge representation schemes (the above three, RLL, and AGE) has been given by Barstow et al. [22]. We will now highlight another such environment which is not detailed there.

LOOPS is a programming environment that integrates four different programming paradigms for the purpose of providing a prototyping system for expert systems [18,19]. The integration has two major themes: combining the four paradigms for use in building expert systems and integration of these paradigms into a programming environment for creating and debugging knowledge-based systems. The four paradigms are: *procedure-oriented programming*, which like most of the familiar programming languages separates data from programs and builds large programs from smaller subroutines; *object-oriented programming*, in which information is organized in terms of objects, combining both data and methods, and in which large objects are built up from smaller ones; *access-oriented programming*, which is used in programs that monitor other programs by attaching *active values* to objects that can themselves trigger other computations when an object's data are accessed; and

rule-oriented programming, which is used to represent the decision-making knowledge in the system and provides (among other features) the mechanisms for explanation and belief revision. The other aspect of the integration is that LOOPS extends many of the facilities available in Interlisp-D [24], such as a display-oriented "break" package, which provides entry into the audit-trail mechanism, as well as various editors and inspectors. For details, see Stefik and Bobrow [18].

We have shown in Fig. 6 an inheritance lattice from a simulation called *Truckin'* which was developed within the LOOPS environment. This would show on a work-station screen as a *class browser,* which is an interactive program that allows one to browse through the knowledge represented there. The lines represent relationships between classes and superclasses, each of which has its own and inherited properties, instances and methods, indicative of the extended frame system discussed earlier. For example, in Fig. 6, we see that a Strawberry is a FragileCommodity, a Fruit and a PerishableCommodity, and one of the Groceries. By accessing each of the nodes in a browser (with a cursor controlled by a three-button *mouse*), the user can access and change the information in this representation. In Fig. 7, we have shown a display of gauges, which are triggered by the active values mentioned earlier. A browser at the bottom of the display shows that the gauges are themselves organized in a lattice and that the DigiScale is a combination of an LCD with an HorizontalScale.

LOOPS was originally designed as a teaching device for disseminating knowledge engineering techniques [19]. However, it has been applied as a prototyping system, for example, to a computer-repair diagnostic task by Bobrow et al. [29], and further extensions and applications are in process.

7 How Is the Knowledge Captured?

We now turn our attention to the process by which the expert's knowledge of his or her domain is elicited and captured for use in a knowledge-based system. It is not a simple, one-stop linear process. In fact, the knowledge engineer and the domain expert will interact over a long period of time, not always continuously, and the expert system will evolve as both parties gain deeper understanding of the task and of different ways to conceptualize and formulate the knowledge required to do the task. Part of the difficulty of capturing the knowledge is that one is not trying to capture fixed algorithmic approaches to problem solving. Rather, the knowledge to be acquired is heuristic, judgmental, subjective, not necessarily codified or organized, and the organization of that knowledge for application is not always consciously understood by the expert him or herself.

Buchanan et al. [30] divide the knowledge acquisition process into five stages: *identification,* in which the important stages of the problem are characterized and goals for the entire project are set; *conceptualization,* during which the key domain-specific attributes of the task are made explicit and some initial thought is given to knowledge representation issues; *formalization,* in which stage a formal model of the task and its key properties and relationships are mapped into a representation scheme; *implementation,* during which the mapping is actually implemented by using whatever tool (skeletal system, programming environment) has been chosen for the project; and *testing,* in which the prototype system is exercised against a library of problems for which solutions are already known. After the testing phase, feedback loops are implemented in which, depending on the outcomes of the testing, the problem may be reformulated, the knowledge representation redesigned, and refinements may be introduced into the knowledge base, as well as into other features of the system.

This conceptual model of the knowledge acquisition process to a large extent hides the interpersonal and institutional aspects of the process of capturing knowledge, as well as those aspects related to successful implementation of an expert system that will be accepted and applied by the intended users. It also hides the issues involved in choosing a suitable task, in building and extending a prototype system, and of keeping a domain expert interested in the process (presumably this is not a problem for the knowledge engineer). These issues have been addressed elsewhere [30,31], but it may be instructive to discuss some of them here.

First of all, the task should be clearly defined, rich in reasoning, and one that is performed by an expert in a reasonable period of time, that is, a period of time measured in hours, rather than in weeks or months. The task itself should be fairly narrow and domain-intensive. In other words, the task ought not to require a lot of general and commonsense knowledge about the world, and the number of relevant concepts should be bounded and not unduly large. An expert must be committed to the project—really committed—for with no expertise to capture there is no knowledge to represent.

The expert should provide a library of case studies or solved problems, for several important reasons. One is that a very effective way for the knowledge engineer to learn about the task is to go through, step

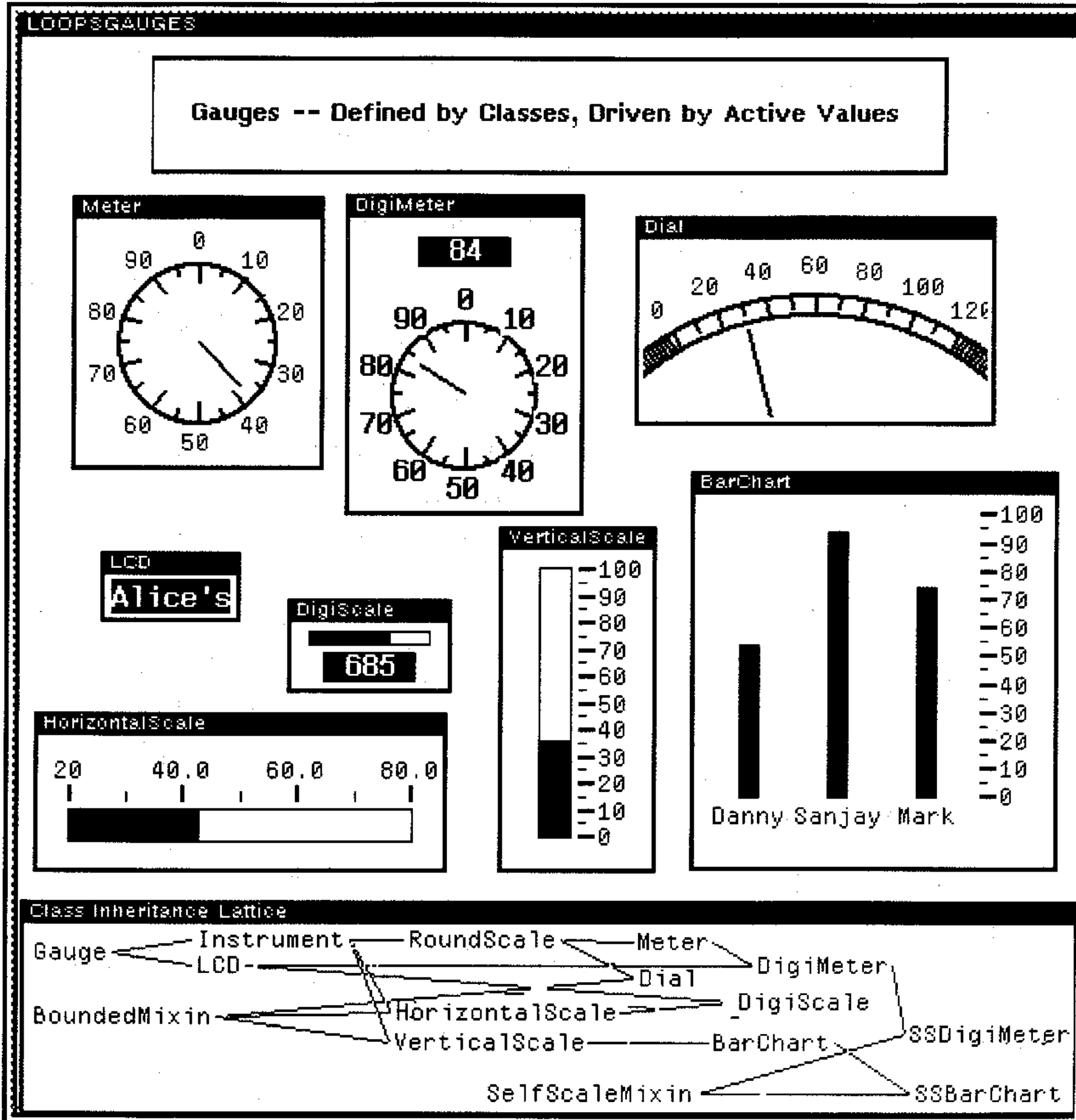

Fig. 7. Gauges in LOOPS (After Stefik et al., Fig. 7 [19])

by step, a solution (or a design or a diagnosis) with the expert in order to help the domain expert make explicit the task and its performance. This will allow the knowledge engineer to assess the plausibility of the project, to see if there is a body of knowledge that can be successfully captured and represented. The examination of a set of cases will also ensure that more knowledge is exposed and captured, for different problems will expose new heuristics that were so "completely obvious" in—or not relevant to—the cases already explicated. Further, the library of case studies will be invaluable in the testing stage, as the system can then be exercised on them and verified against the known results.

The domain expert must be interested in the project, for he will have to devote a lot of time to it. The process of eliciting his or her knowledge will take many interviews and interactions, and these will typically occur over a long period of time. In fact, the system itself will probably go through several itera-

tions, and more likely, the first system (the "Mark-I" version) will be thrown away altogether. One of the reasons that several iterative cycles will be traversed, providing at the same time an important benefit for the expert, is that the expert gains an increasingly sharper and deeper understanding of his own expertise as the knowledge acquisition process unfolds. It is probably a good idea to consider the entire process experimental and to expect that the Mark-I system will be thrown away after 3 to 6 months and that a final system may not be completed for a year or two. The initial development of SACON, which was built on a known skeletal system with an appropriate rule-based structure [6], required 2 man-months of the expert's time to make explicit the consultation task and to formulate the knowledge base, and another 2 man-months to implement and test the rules. The expert system R1 required more than a year of development, refinement, validation, and testing—involving a lot of people—before the Digital Equipment Corporation decided that it was sufficiently reliable to be used on a regular basis [31]. The integration of R1 into Digital's organizational structure also took more than a year to plan and implement [31].

Obviously, much more can be said on this subject than time and space presently permit. Perhaps the most important point is that the construction and implementation of an expert system, especially the process of eliciting, understanding, and representing knowledge, is still very much an experimental process. Although there are increasing numbers of successful systems and of knowledge programming environments, we are still a very long way from the maturity of more traditional computational fields, such as finite element analysis.

8 On Some Cultural Aspects of Knowledge Engineering

What do we mean when we use the phrase "knowledge engineering"? As previously outlined, the knowledge engineer elicits, represents, and captures knowledge so that it can be used, shared, and even extended to create new knowledge. Thus, the knowledge engineer is actually in the business of *engineering knowledge*. It is an interesting notion, that one can shape knowledge for different purposes, and although it is not entirely a new notion—certainly any teacher must think about making a subject understandable and putting it into context—new technology gives this concept much greater scope than it ever could have had. The new technology will encompass more than expert systems; it includes symbolic programming, networking, distributed computing, powerful interface tools and programming environments, parallel computing, and more. It reflects new ways of looking at computers and computation, and at the reasons we use computers. It allows, even encourages, new ways of looking at the generation and diffusion of knowledge. The effect on our professional lives and environments is potentially so dramatic that a few thoughts on the broader context of knowledge engineering are in order. Of particular interest are the possibilities enabled by networking, knowledge-based systems, and their interaction.

The goal of the activities described in this paper is the construction of a computer program, a program that replicates the symbolic and heuristic reasoning of an expert, but nonetheless a computer program. But there is more to knowledge engineering than computer programs, for expert systems and AI techniques offer a new and *active medium* in which to manipulate knowledge [32]. That we can elicit, represent and extend knowledge suggests some very interesting questions [32]: From where does knowledge come? What are its measures? How is knowledge generated and propagated? Can differences in social and technical infrastructure change the way knowledge is generated, propagated, and integrated into new environments? The last question ought to be of particular interest to an engineering and design community so fragmented as ours, and the telling of a particular tale will be quite instructive here.

In the mid-1970s a collaborative effort was undertaken between two groups to search for better methods with which to do VLSI design. One group was headed by Lynn Conway at Xerox PARC, the other by Carver Mead at Caltech. By 1977, some important basic results had been achieved, and so the issue of propagating the new design methodologies was addressed. As Conway put it [33], the collaborators were quite "aware of the difficulty of bringing forth a new system of technology by just publishing bits and pieces of it among traditional work." Conway suggested that instead of *writing* a book, a very traditional academic approach to knowledge dissemination, they guide the *evolution* of a book through an interactive computer network, working together with interested faculty and students at a few universities. The network interaction was done through the ARPAnet, a network that links computer research groups and facilities at universities, research laboratories, and government laboratories and agencies.

The text material and design methods were initially debugged with a relatively small number of participants. By the summer of 1978 a draft of the entire manuscript was finished, that draft incorporating a text, a course curriculum, and a coherent set of design

methods. Attention then turned to the question of wider diffusion, of greater portability. So the network experiment became a "network adventure" which involved many more schools, faculty, and students. As an incentive for their participation in this adventure, participants were offered the chance to have their VLSI designs implemented on a chip, the *MultiProject Chip* (*MPC*), if they used design rules from the Mead-Conway text and submitted their designs in a specified protocol. This approach allowed students to validate their designs and Mead and Conway to achieve widespread propagation of their design methods. By 1980, the Mead-Conway design methods were setting the standards for VLSI design in the United States [34]. The network interactions allowed a rapid diffusion of knowledge and the standardization of design protocols. Such networks have high branching ratios, short time constants, and flexibility of social structure [33]. Anyone on the net can broadcast a message that will reach all the others on that distribution list, and very quickly. As Conway notes, a highly interactive electronic network allows one to "modify the operation of a large, experimental, multiperson, social-technical system while it is under test." Further, as evidenced by the chip design implementation strategy adopted in the MPC adventure, access to an attractive service allowed the relatively straightforward adoption, by an entire community, of a uniform set of standards.

Now, the VLSI design methodology propagated in the MPC adventure was not one based on expert systems, although there is ongoing work in that area [2]. But an interesting point of intersection between expert systems and the network interactions just described is embodied in the recent development of an expert diagnostic system called DARN by Bobrow et al. [29]. That intersection is inherent in the view that the expert system DARN is viewed as a community memory, that is, as a collective repository of experience and wisdom, some heuristic and some factual, relating to the repair-oriented diagnostics for a particular model of work-station. Here the expertise captured belongs to a community of experts, and it can be accessed and updated by all the members. In fact, DARN is also used as a training device for technicians new to the repair group, as they are able to learn a lot about their domain from exercising the system themselves. Thus, we see that expert systems can also be used to *propagate* knowledge.

Perhaps what is most important about this context is that many of the ideas originating in AI and knowledge engineering appear to be quite consonant with the direction in which computation is headed [1]. It is not that we are done with number-crunching or other algorithmic approaches to programming. But the next generation of computers, the so-called fifth generation, is likely to have two principal features. It will do parallel processing (in arrays of processors), rather than sequential processing of its instructions, and it will work in a symbolic language like Lisp or Prolog, rather than in the general-purpose, computationally oriented languages with which we are all familiar. The metric of speed may no longer be the number of instructions performed per second rather, it may be (or is, at least for the Japanese project [1]) *lips*, that is, logical inferences per second. But what is most important about these machines, which the Japanese call knowledge information-processing systems (*KIPS*), is that the style of programming them and interacting with them will be very different from our current interactions with computers. Rather than tell the machines how to do what we want them to do, as we do now, the Japanese would have us state our intentions and let the computer—the KIPS—figure out how to implement the given task. This suggests, as will become clearer in the next section, that as engineers we should begin to pay a lot of attention to how we structure our work and exchange ideas, for we can use some of these notions very effectively to do better engineering.

9 Roles for Expert Systems in Engineering

In this section we turn to potential applications of expert systems to engineering. To date, most of the developments in expert systems have been in fields other than engineering, but there are a few examples to which we can point. First of all, there is the SACON system described earlier [6,7], and a similar system written in Fortran was described by Rivlin et al. [35]. There are some diagnostic systems, which may be considered very practical engineering, including the DARN system mentioned earlier [29] and a similar fault-finding troubleshooter for locomotive repair, called CATS-1, which was developed by the General Electric Company [36]. Some applications in the field of civil engineering include: HYDRO, an expert system that assists a hydrologist in estimating input parameters to a (algorithmic) watershed simulation program [37]; SPERIL1, a system to assess seismic damage to buildings [38]; and HI-RISE, a system being developed to do preliminary design of high-rise buildings [39]. In the field of mechanical engineering, there have been calls for applications of knowledge-based systems to engineering design [40–42], but no working systems have been reported. There has been some success in developing systems for VLSI design [21,43,44]. Finally, one small instructional system has been reported [47].

So where do we go from here? One start is to recapitulate the kinds of tasks that expert systems have been designed to perform and see if we can identify engineering tasks to match. The expert system tasks are [5]: *interpretation,* the analysis of data to determine their meaning; *diagnosis,* the process of fault identification in a system; *monitoring,* the continuous interpretation of signals, coupled with the triggering of warnings if intervention is required; *prediction,* the forecasting of future states from existing models; *instruction,* the diagnosis and repair of student understanding; *planning,* the preparation of a program of activities to achieve a set of goals; and *design,* the identification of specifications and requirements to create new objects. Note that of the engineering systems discussed earlier, four are tasks of diagnosis and consultation [6,29,35–38]; three are concerned with VLSI design [21,43,44] and one with structural design [39]; and one is concerned with instruction [47].

The tasks of interpretation, diagnosis, and monitoring are especially ripe for application to engineering problems. Each of these requires the ability to obtain data and sift it in ways that require experienced, reasoned, and often heuristic judgement. Further, each of these tasks could probably use a rule-based representation and a backward-chaining inference engine, each of whose organizations are amply tested and well understood. Across a range of concerns in aeronautical, civil, and mechanical engineering, for example, it is easy to generate a list of potential tasks (see also Fenves [46]):

- assessment of fatigue and wear in structures
- monitoring and diagnosis for rotating machinery
- monitoring of physical degradation of infrastructure elements such as piping networks and roads
- failure assessment for structures and soils
- choosing among modeling and analysis tools
- evaluation of designs

In many of these applications, the expert system will have to be linked to some transducers and signal processors to provide the factual input about the state of the world in which the task is being performed. However, it is likely that inexpensive sensors and microprocessors will make such monitoring both routine and extremely useful, when coupled with an expert system.

Tasks in planning and design are harder, but they are probably more important and often more interesting. They are harder because the solution space is usually very large and the solution needs to be formed or synthesized. Also, the representation of design constraints and the choices for pruning in planning and design searches are more difficult to implement than are the comparable issues in diagnostic tasks, for example. Some potential tasks across the span of engineering disciplines mentioned previously could include (see also [46]):

- environmental assessment for proposed designs of structures, roads, and other infrastructure elements
- selection of structural configuration for preliminary design
- planning for maintenance and rehabilitation of machinery, infrastructure, and other objects that wear and degrade over time
- configuration of interactive devices in constrained spaces, such as the layout of all the parts in a copying machine or the distribution of equipment and materials around a construction site

Planning and design are central to the engineering function, and so many more examples will readily occur to any engineer.

All the tasks just outlined, and more, meet the requirements for their appropriate capture in expert systems, assuming that there are experts willing to participate. That is, for each of these generic tasks, a narrow, domain-specific set of questions can be identified, the answers to which require reasoned experience. Further, these tasks occur often, providing both case studies and a source of leverage.

Viable tasks will also be identified in engineering education, where two broad areas of application come to mind. The first could simply be an extended notion of computer-aided instruction or of self-paced learning, where the student is guided through a complex field by an expert system. In this model, educational software will be designed, using expert system technology, to guide a student through a set of exercises, say, help the student find and diagnose errors in his solutions, and then formulate better solution approaches. This kind of software can be used to expose the student to more complex and practical problems than can be done with standard textbook problems, and it can be used to encourage reasoning and heuristic styles of problem solving [47]. The other model for enhancing engineering education is basically that of using expert systems as pre- and postprocessors for more standard computer codes. Here the expert system will help the student identify the correct modeling approach and input, verify, and validate the output, and ascertain that the solution is consistent with the assumptions inherent to the model used. This approach can obviously be used in practice as well, enabling a less experienced engineer to use properly sophisticated number-crunching software that was developed elsewhere, as in the SACON system [6,7].

The preceding lists of potential uses for expert systems are more suggestive than exhaustive, and there will be influences from some of the other tech-

nologies mentioned in passing. For example, the advent and integration of highly interactive programming environments, expert systems, and interactive networks could facilitate the rapid propagation of design techniques and manufacturing standards. It is perhaps more difficult to envision this on a national scale, where design and manufacturing environments are dispersed over a large country in a fragmented engineering and design community. It is a task that is intimidating even for a single large company, if its product lines are diverse and its facilities dispersed. On the other hand, the success of the MPC adventure at least suggests that there are gains to be made [33]. And certainly it is unarguable that better tools for integrated design and manufacturing, and the rapid and widespread adoption of uniform standards, could both lead to significant gains in productivity that are badly needed today in many segments of our economy.

10 Conclusions—and a Few Cautions

The title of this last section presages the fact that we will conclude this introduction to expert systems and knowledge engineering on a cautionary note. There is no doubt that the potential for dramatic, positive impact is there. As noted, the utility of interactive and intelligent computer systems might well be unbounded. However, one cannot generally proceed simply by learning Lisp, hiring an expert, and writing a set of rules. First of all, as noted, the knowledge for the task may not be suitably represented in production rules, and there may not be a committed expert who is willing to share his hard-earned expertise. But there are other reasons for restraint.

The task chosen for representation must be reasonably narrow, performed fairly often, and have a cadre of experts who agree in general terms on appropriate approaches. Expert systems are not meant for ill-defined tasks, nor are they effective where common sense and general knowledge are required. As Buchanan et al. [30] have noted, "today's expert systems fall well short on dimensions requiring general intelligent behavior. They are more akin to *idiots savants* than to real human experts." Further, even if the task is appropriate, the technology for representation and reasoning may not be sufficiently well understood or available. There are, for example, better facilities for dealing with diagnosis and interpretation than there are for planning and design. This fact alone should make the engineering community somewhat wary.

The resources required to build a robust and significant system are also substantial. Recall that SACON required that the domain expert alone invest 4 man-months in its development, that time *not* including "the necessary time devoted to meetings, problem formulation, demonstrations and report writing" [7]. That estimate does not account for the time devoted by the knowledge engineers or any other staff associated with the project. And SACON is a relatively small system, with a sharply defined and clearly stated task, and built on a representational skeleton that is virtually ideal for that task. Expert systems are expensive, requiring significant investments of human and capital resources.

It is also important that expectations for expert systems be kept within manageable bounds. This means that we should not make grand promises for any particular project, both as to utility and as to cost. It also means that we should not expect this technology to overwhelm the engineering workplace overnight. There is a shortage of people trained in AI and knowledge engineering, and it is not a technology that newcomers will easily absorb without significant training and experience. Indeed, the current shortage of trained talent in knowledge engineering means that there is also a shortage of skilled teachers who can contribute to the diffusion of this knowledge. The number of active research and educational programs in this area is relatively small (see, for example, Appendix C of [1]). This situation has been exacerbated by the sudden appearance of several start-up companies in expert systems, the equally sudden profusion of corporate-sponsored AI laboratories, and the announcement of a large strategic computing program by the U.S. Department of Defense, in which knowledge engineering and related expertise will play significant roles [48].

Nonetheless, all the preceding reservations notwithstanding, it seems clear that expert systems and knowledge engineering, including networking and highly interactive programming environments, will have noticeable and positive effects on the conduct of engineering practice. Even more than the advent of high-speed algorithmic computing and the development of extremely powerful graphics tools, the new technologies discussed in this paper offer major opportunities for understanding and doing good engineering. It is a large and daunting vision, but an exciting one.

Acknowledgments

I wish to acknowledge personal debts to two colleagues who have made possible my own entry into this exciting new field. They are, first, Steven J. Fenves, University Professor at Carnegie-Mellon University, who introduced me to this activity with an invited luncheon talk just 2 years ago [45] and has encouraged my interest since; and second, Mark J. Stefik, Principal Scientist and Manager of the Knowledge Systems Area at PARC, who has provided a lot of

support in many ways and has immeasurably contributed to my understanding of expert systems and its broader context, knowledge engineering. I would also like to thank several colleagues (all at Xerox PARC but for the exception noted) who reviewed and commented on drafts of this paper: Daniel G. Bobrow, Johan de Kleer, James W. Male (University of Massachusetts and Stanford University), Sanjay Mittal, Mark J. Stefik, and especially Dan S. Bloomberg, who read many, many partial drafts and iterations. Finally, I wish to acknowledge generous financial support by the Xerox Palo Alto Research Center, whose grant to the University of Massachusetts made possible my sabbatical year at PARC.

References

1. Feigenbaum, E.A.; McCorduck, P. (1983) The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World. Reading, MA: Addison-Wesley
2. Stefik, M.J.; Conway, L. (1982) The principled engineering of knowledge. AI Magazine, 3(3), 4–16
3. Brachman, R.J. (1983) What are expert systems? In: Building Expert Systems (Eds. F. Hayes-Roth; D.A. Waterman; D.B. Lenat). Reading, MA: Addison-Wesley
4. Hart, P. (1982) Directions for AI in the eighties. SIGART Newsl. (79), 11–16
5. Stefik, M.J. et al. (1983) Basic concepts for building expert systems. In: Building Expert Systems (Eds. F. Hayes-Roth; D.A. Waterman; D.B. Lenat). Reading, MA: Addison-Wesley
6. Bennett, J.S.; Englemore, R.S. (1979) SACON: A knowledge-based consultant for structural analysis. In: Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo
7. Bennett, J.S. et al. (1978) SACON: A Knowledge-Based Consultant for Structural Analysis. Stanford Heuristic Programming Project Memo HPP-78-23, Stanford University, Palo Alto, CA
8. Shortliffe, E.H. (1976) Computer-based Medical Consultation: MYCIN. New York: American Elsevier
9. Nii, P. as quoted in ref. 1, p. 83
10. Stefik, M.J. Personal communication
11. Charniak, E.; Reisbeck, C.K.; McDermott, D.V. (1979) Artificial Intelligence Programming. Hillsdale, NJ: Erlbaum
12. van Melle, W. (1979) A domain-independent production-rule system for consultation programs. In: Proceedings of the Sixth International Joint Conference on Artifical Intelligence, Tokyo
13. McDermott, J. (1980) R1: An expert in the computer systems domain. In: Proceedings of the First Annual National Conference on Artificial Intelligence, Stanford University, Palo Alto, CA
14. Stefik, M.J. et al. (1983) The architecture of expert systems. In: Building Expert Systems (Eds. F. Hayes-Roth; D.A. Waterman; D.B. Lenat). Reading, MA: Addison-Wesley
15. Duda, R.O.; Shortliffe, E.H. (1983) Expert systems research. Science 220(4594), 261–268
16. Barr, A.; Davidson, J. (Eds.) (1981) Knowledge representation. In: The Handbook of Artificial Intelligence, Vol. I (Eds. A. Barr; E.A. Feigenbaum). Los Altos, CA: Kaufmann
17. Woods, W.A. (1975) What's in a link: Foundations for semantic networks. In: Representation and Understanding: Studies in Cognitive Science (Eds. D.G. Bobrow; A. Collins). New York: Academic Press
18. Stefik, M.J.; Bobrow, D.G. Four paradigms of programming, manuscript in preparation
19. Stefik, M.J. et al. (1983) Knowledge programming in LOOPS: Report on an experimental course. AI Magazine 4(3), 3–13
20. Kinnucan, P. (1984) Computers that think like experts. High Tech. 4(1), 30–41
21. Stefik, M.J.; de Kleer, J. (1983) Prospects for expert systems in CAD. Comput. Des. 22(5), 65–76
22. Barstow, D.R. et al. (1983) Languages and tools for knowledge engineering. In: Building Expert Systems (Eds. F. Hayes-Roth; D.A. Waterman; D.B. Lenat). Reading, MA: Addison-Wesley
23. Waterman, D.A.; Hayes-Roth, F. (1983) An investigation of tools for building expert systems. In: Building Expert Systems (Eds. F. Hayes-Roth; D.A. Waterman; D.B. Lenat). Reading, MA: Addison-Wesley
24. Anon. (1983) Interlisp Reference Manual. Xerox Special Information Systems, Pasadena, CA
25. Moon, D.A. (1974) Maclisp Reference Manual. Massachusetts Institute of Technology, Cambridge, MA
26. Sheil, B. (1983) Power tools for programmers. Datamation 29(2), 131–144
27. Clocksin, W.F.; Mellish, C.S. (1981) Programming in Prolog. New York: Springer-Verlag
28. Goldstein, I. as quoted in ref. 20, p. 40
29. Bobrow, D.G.; de Kleer, J.; Mittal, S. DARN: A community memory for a diagnosis and repair task, manuscript in preparation
30. Buchanan, B.G. et al. (1983) Constructing an expert system. In: Building Expert Systems (Eds. F. Hayes-Roth; D.A. Waterman; D.B. Lenat). Reading, MA: Addison-Wesley
31. McDermott, J. (1981) R1: The formative years. AI Magazine 2(2), 21–29
32. Stefik, M.J.; Conway, L. manuscript in preparation
33. Conway, L. (1981) The MPC Adventures: Experiences with the Generation of VLSI Design and Implementation Methodologies. Xerox Palo Alto Research Center, Palo Alto, CA
34. Mead, C.; Conway, L. (1980) Introduction to VLSI Systems. Reading, MA: Addison-Wesley
35. Rivlin, J.M.; Hsu, M.B.; Marcal, P.V. (1980) Knowledge Based Consultation for Finite Element Structural Analysis. U.S. Air Force Flight Dynamics Laboratory Report AFWAL-TR-80-3069, Wright-Patterson Air Force Base, OH
36. As cited in ref. 20, p. 37
37. Gaschnig, J.; Reboh, R.; Reiter, J. (1981) Development of a Knowledge-based System for Water Resources Problems. SRI International Technical Report 1619, Menlo Park, CA
38. Ishizuka, M.; Fu, K.S.; Yao, J.T.P. (1981) SPERIL 1-Computer Based Structural Assessment System. Purdue University Technical Report CE-STR-81-36, West Lafayette, IN
39. Fenves, S.J.; Maher, M.L. (1983) HI-RISE: An Expert System for the Preliminary Structural Design of High Rise Buildings. Department of Civil Engineering Technical Report, Carnegie-Mellon University, Pittsburgh, PA
40. Dixon, J.R.; Simmons, M.K. (1983) Computers that design: Expert systems for mechanical engineers. CIME 2(3), 10–18
41. Brown, D.C.; Chandrasekaran, B. (1983) An approach to expert systems for mechanical design. In: Proceedings, Trends and Applications, National Bureau of Standards, Gaithersburg, MD
42. Elias, A.L. (1983) Computer-aided engineering: The AI connection. Astronaut. Aeronaut. 21(7-8), 48–54
43. Mitchell, T.M. et al. (1983) An intelligent aide for circuit redesign. In: Proceedings of the Fourth National Conference on Artificial Intelligence, Washington, DC
44. Rychener, M.D. (1983) Expert systems for engineering design: Experiments with basic techniques. In: Proceedings of the IEEE Design Automation Conference, Gaithersburg, MD

45. Fenves, S.J. (1982) New uses of computers: Knowledge-based systems. MIT Workshop on Microcomputers in Civil Engineering, Massachusetts Institute of Technology, Cambridge, MA
46. Fenves, S.J. (1983) Artificial intelligence-based methods for infrastructure evaluation and repair. New York Conference on the Infrastructure, New York
47. Starfield, A.M. et al. (1983) Mastering engineering concepts by building an expert system. Eng. Ed. 74(2), 104–107
48. Anon. (1983) Strategic Computing: New-generation Computing Technology: A Strategic Plan for Its Development and Application to Critical Problems in Defense. Defense Advanced Research Projects Agency, Arlington, VA