1-1-1987

Issues in the Design and Implementation of Expert Systems

Clive L. Dym
Harvey Mudd College

Issues in the design and implementation of expert systems

Clive L. Dym

Artificial Intelligence for Engineering, Design, Analysis and Manufacturing / Volume 1 / Issue 01 / February 1987, pp 37 - 46
DOI: 10.1017/S0890060400000135, Published online: 27 February 2009

(*AI EDAM*) (1987) 1(1), 37–46

ISSUES IN THE DESIGN AND IMPLEMENTATION OF EXPERT SYSTEMS[1]

CLIVE L. DYM

Department of Civil Engineering, University of Massachusetts at Amherst, Amherst, MA 01003, U.S.A.

This article discusses the issues that arise in the design and implementation of expert systems. These issues include: task selection; the stages of development of expert system projects; knowledge acquisition; languages and tools; development and run-time environments; and organizational and institutional issues. The article closes with some speculation about the future development of expert systems.

1. Introduction

This article is concerned with the *process* of building an expert system and, as such, its focus is on the questions of *how* such a system is implemented or built. For the purpose of this discussion, an expert (or knowledge-based) system is one that uses knowledge captured from experts to solve problems or do tasks that involve reasoning. Such systems differ from conventional algorithmic programs in that they incorporate heuristics and other rule-oriented reasoning aids, and their architectures typically represent a separation of the control of the program from the knowledge of the domain (Dym 1985*a*). The tenor of the discussion is discursive and tutorial, and pointers to some of the salient literature are given. Among the issues addressed that arise from implementation of a system are the following:

What tasks are suitable for encapsulation within an expert system? What are the stages of development of an expert system?

How is knowledge acquired? Can more than one expert be used? How are the experts interrogated? How is conflict (among several experts) resolved?

What are the technical issues? What are the appropriate languages and tools for developing a serious system? What are the appropriate languages and tools for running or delivering a rapid and efficient system? How do tools and languages relate to hardware?

What are the issues for the organizations that undertake the building of expert systems? Are expert systems expensive? How is access to the experts guaranteed? How important is the commitment of management? What are the advantages (i.e. community memory, standardization, training and eduction) to management?

The article concludes with some speculation on future issues in the building of expert systems, especially those that appear to be affected by the advent of parallel architectures. The most intriguing prospect here is the possibility of successful interaction of multiple expert systems acting concurrently.

Finally, it is again noted that the treatment of these issues is discursive and somewhat idiosyncratic, heavily conditioned by the author's own experience in this field (Dixon and Dym, 1986, Dym, 1985*a*, 1985*b*, 1987, Mittal and Dym, 1985, Mittal *et al.*, 1986, Morjaria *et al.*, 1985).

2. Task selection and the stages of development

Careful selection of the task to be encapsulated in an expert system is essential for the success of the system development project. Several criteria for choosing projects and tasks have been enumerated (Bobrow *et al.*, 1986, Dym, 1985*a*, Prerau, 1985) and will be summarized here. It will subsequently be seen that the winnowing of a list of candidate projects according to these criteria is the first stage ('identification') in the development of an expert system.

The task to be modelled in an expert system must be clearly defined, rich in reasoning, and one that is performed by an expert reasonably often and within a

Received 3 February 1987.

[1] To appear in revised form as Chapter 3 of M. L. Maher (Ed.), *Expert Systems for Civil Engineering: Technology and Applications*, New York: American Society of Civil Engineers, 1987.

0890-0604/87/010037 + 10$02.00/0

reasonable period of time. The task itself should be fairly narrow and domain-intensive. It ought not to depend on a lot of general and commonsense knowledge about the world, and the number of relevant concepts should be bounded and not unduly large. If it is not rich in reasoning, and if more specifically the task is capable of algorithmic solution, it is not appropriate for expert system encapsulation.

What is a reasonable length for the time it takes to do a task that will be captured in an expert system? The conventional wisdom has it that the time should be measured in hours, rather than in weeks or months. It appears, however, that this is too simple an answer. In fact, there are some tasks that take weeks or months that require a lot of repetitive and tedious work that can be successfully modelled in an expert system (Dym *et al.*, 1987). The real issue is the underlying complexity of the task and the degree to which it involves much creative thought. If a task requires days or weeks of original and creative thought by an expert, then it is more than likely not suitable for encapsulation.

There should be available a substantial library of case studies or test cases. This library is useful in the knowledge acquisition process (see Section 3), for testing implementations of the system, and for confirming that the criteria for task performance time and frequency have indeed been met. Test cases are not readily produced if the task is performed only sporadically, and a lack of repeatability or task consistency is also signalled by a paucity of case studies. The examination of a set of cases also helps ensure that more of the knowledge is exposed, for different examples expose new heuristics that were so 'completely obvious' in—or not relevant to—the cases already examined.

The task must have clear—and typically economic—value to the organization sponsoring the development of an expert system because, as will be seen in greater detail below, the building of an expert system is expensive and time consuming. Thus, the expert system must be capable of producing substantial benefit for the sponsor to justify the committment of sufficient resources to insure a successful outcome. A particularly interesting example of this is the R1/XCON system developed by the Digital Equipment Corporation (DEC) which will be discussed in somewhat greater detail in Section 5 (McDermott, 1981).

The selection of a task for coding in an expert system is the first step in building an expert system, and in fact it may be viewed as the beginning of the first of the five stages of system building (Buchanan *et al.*, 1983). These stages can be characterized as follows: *identification*, in which the important stages of the problem are characterized and goals are set for the entire project; *conceptualization*, during which the key attributes of the task and its domain are made explicit and some thought is given to issues of knowledge representation; *formalization*, at which time a formal model of the task, its attributes and their relationships are represented in a particular scheme (or set of schemes); *implementation*, during which time the representation scheme is programmed using whatever tool (programming language, shell, etc.) has been chosen for the project; and *testing*, in which the prototype system just implemented is exercised against some of the case studies from the previously assembled library of solutions. After the testing phase, or rather, within the testing phase, feedback loops are implemented which will lead to further refinement of the prototype or even a complete reformulation.

This conceptual model of how knowledge is acquired, represented and coded hides to a large extent the institutional and personal aspects of capturing knowledge, as well as aspects of implementing a system that will be used by its intended audience and appreciated by its sponsors. This model also hides the role that experts play in the design and implementation of an expert system, and how important it is to maintain their enthusiastic participation throughout the life of the project. Some of these aspects are addressed in greater detail below, especially in Sections 3 and 5.

3. Knowledge acquisition

This section is devoted to a discussion of how knowledge is acquired from the expert(s), an essential step if there is to be an expert system! This part of the system-building task transcends the first three stages (identification, conceptualization and formalization) in the paradigm outlined in the previous section. The present discussion begins by focussing on what is required in the way of knowledge. There then follows a model of how the knowledge is obtained from the experts themselves. Note that there will be continued reference to multiple experts, reflecting in part the author's strong bias that several experts ought to be consulted in the process of building an expert system (Mittal and Dym, 1985). In addition, it should be kept in mind that other sources of expertise can be exploited, including analysis programs, books, research papers, reference manuals, etc.

Knowledge acquisition begins with choosing and/or defining the task that will be encapsulated within the

expert system. The choice, as noted earlier, depends on several critical factors, including the time taken to perform the task, its value, the availability of experts, the reasoning involved, and so on. This phase of the project also marks the formal entry of the *knowledge engineer,* i.e. the expert in AI and/or computer science who will lead the knowledge acquisition and implementation phases of the expert system development. (The knowledge engineers are distinct from the *domain experts*, commonly referred to simply as the experts, whose knowledge and expertise about the domain is being put into the expert system.) As the knowledge engineers being to familiarize themselves with the task at hand, they will begin to develop a domain vocabulary. That is, they will begin to acquire and use the words, phrases, formulas, etc., which comprise the natural language in which the task is described. This is an important step in understanding the domain and (later) in communicating with the experts.

With the vocabulary in hand, the knowledge engineer begins to sort out how the task is performed by developing a model of the reasoning involved and how it is applied. This is an intellectural exercise that may be represented by a flow chart or by steps in a protocol or so on; it is not (yet) a computer program or even a formal representation. It is, rather, a paper exercise in which the task is outlined and modelled so that it can be evaluated for encapsulation. Of course, the process of laying out the task often provides the initial ideas about the subsequent representation of the task, but that is a later step, one taken (typically) by the knowledge engineers acting separately from the domain experts. But, in terms of the domain knowledge itself, the end of the acquisition process is marked by the completion of a document, flow chart, etc., that traces the task in its natural language and makes explicit what knowledge is used, where, when, and how a conclusion is reached.

The process of acquiring knowledge is an interactive one representing a complex dialogue between the domain experts and the knowledge engineers. Many questions arise in such a process, including: How many experts are needed? Is one expert enough? How are experts to be identified? How does one verify an expert's expertise? Can a domain expert also be his/her own knowledge engineer? How are experts interviewed? How are case studies used in the process? Are there ways to resolve conflicts among experts?

There is not sufficient space here to answer all these (and related) questions. Rather, the approach taken will be to set forth the strategy taken in the PRIDE project (Mittal *et al.,* 1986), detailing along the way the rationales and strengths of this particular strategy (Mittal and Dym, 1985). The story picks up after an agreement-in-principle had been reached between the project sponsors (Xerox's Reprographics Business Group in Webster, NY) and the knowledge engineers (both located at the Xerox Palo Alto Research Center) to build an expert system to aid in the subsystem design of paper copiers. This particular expert system would be built as a demonstration that the technology could be used successfully in an engineering environment in order to carry out better engineering designs.

There were two designers within the sponsoring organization who would work closely with the knowledge engineers on this project. Through them, and following the preliminary selection of a task for encapsulation, in accord with the criteria set out in the previous section, the knowledge engineers for the PRIDE project arranged a set of interviews with a panel of experts who worked within the sponsoring organization. The initial motivation for this first set of interviews was to confirm that there was a discrete, repeatable, capturable task that might be worth encoding. This was accomplished by keeping the two designers as resident experts on the knowledge engineers' side of the table, while individually interviewing more than half a dozen other designers who performed the task. The resident experts posed the design problem to each of the other experts in turn, acting as interlocutors and framing the problem exactly as it would be handed to designers in practice, with the experts then working out the design solution at a blackboard.

This process produced some immediate dividends. It became clear that all the experts followed a very similar strategy for carrying out the design in terms of how they decomposed the problem into subproblems, worked on the subproblems, and then related the partial designs to each other. While decomposition is not a new technique for solving complex problems, it was interesting and reassuring to see the similarities in the subproblems chosen and in the strategies for solving each subproblem. Thus, the task chosen seemed to exhibit some regularity that each expert was trained to exploit. Further, it also became clear that there were interesting individual nuances that emerged in each design, reflecting the varying backgrounds of the individual designers, so that the final assemblage of knowledge might benefit from bringing together these separate strands of expertise.

A very important point needs mentioning here. The design problem was posed to the experts in standard form, together with the instruction that they proceed as they would normally to achieve a successful design.

Thus, they were asked to *do* the design; they were *not* asked to describe how they did it. This is important because experts typically cannot give reliable accounts of their expertise and how it is exercised. To extract knowledge, it must be exercised on problems.

Again, the present forum does not allow a complete discussion of all facets of knowledge acquisition. The salient points of the experience described here are that: more than one expert can be used to reinforce the understanding of a repeatable and capturable task; that individual expertise can be captured in such a way as to augment the knowledge of all the users of an expert system [see the discussion of community memory in (Dym, 1985*a*) and (Mittal and Dym, 1985)]; and that experts should be asked to perform their task, rather than describe it.

It is also worth adding that the library of case studies emerges as an important part of the acquisition process, especially where more than one expert is involved. With more experts participating, there will generally be more case studies available and thus there are more opportunities to test cases that are new to particular experts. This facilitates the exploration of task regularity and the emergence of subtle features and points that are not well understood, and it helps highlight areas of conflict. Again, further details are given elsewhere (Mittal and Dym, 1985).

4. Development and run-time programming environments

AI LANGUAGES AND ENVIRONMENTS

Implementation is the next stage in the development of an expert system, where programming of the task is undertaken within a chosen representation scheme. Applications projects in AI are usually implemented in a high-level programming language which can be considered to fall within one of three groupings (Hayes-Roth *et al.*, 1983): general purpose programming languages; general purpose representation languages; and domain-independent expert system frameworks. Some very brief descriptions and examples follow; comprehensive reference lists are not included here as they are available elsewhere (Harmon and King, 1985, Hayes-Roth *et al.*, 1983, Maher *et al.*, 1984, Mullarkey, 1987, Rehak and Fenves, 1985, Waterman, 1985).

The *general purpose programming languages* used in AI are, typically, LISP and its dialects, especially in the U.S., and PROLOG, largely in Japan and Europe. LISP is an acronym for the list processing language developed by John McCarthy in the 1950s, whereas PROLOG is based on the predicate calculus and was developed in Europe in the 1970s. These high-level languages are quite often exercised in *exploratory programming environments* that encourage experimentation with large chunks of knowledge, abstraction, symbolic manipulation, and tentative modifications. Some of the advantages of these environments are outlined by Sheil (1983).

LISP has been the major force behind virtually all the major AI innovations and the impetus for the architectural design of symbolic processors. PROLOG, while popular in Europe, has not been accepted by the majority of AI researchers. PROLOG does express knowledge in logical form, but it has a single type of control structure that many believe is a critical weakness.

There are many dialects of LISP, and names such as Interlisp-D, Zeta LISP and MACLISP will be encountered in the literature. However, Common LISP has become the standard dialect because the DOD funding agencies, and DARPA in particular, have led the push toward a standardization of programming languages.

It is worth repeating that the reason that these LISP environments are so important is that they provide a powerful set of software tools that tremendously enhance programmer productivity (Sheil, 1983). These development tools include code and memory management, editing and debugging, and the ability to handle (and abstract from) large chunks of knowledge.

There are also software systems that facilitate the task of building large expert systems by providing the programmer with a high-level framework, as well as editing and debugging tools. These systems, or environments, are built on top of LISP environments, and include general purpose representation language environments and expert system frameworks.

General purpose representation languages are programming languages designed specifically for knowledge engineering. They are not tied to any particular control or inference strategy, but they facilitate the implementation of a range of problems along the classification-formulation continuum. Such languages include SRL, RRL, KEE, OPS5, ROSIE, ART, LOOPS and AGE.

Domain-independent expert system frameworks, usually referred to as 'shells', provide an inference mechanism, which can then be applied by the addition of domain-specific knowledge. These systems often provide modules for knowledge acquisition and for explanation. They often have their roots in specific expert system projects, from which they thus derive their control strategies. Examples of these expert

system shells include EMYCIN (which was used in the SACON experiment), KAS, HEARSAY-III, EXPERT, and KMS/KES.

Paranthetically, it is worth noting that the term 'shell' is sometimes used to include general purpose representation languages (e.g. KEE, LOOPS) as well as expert system frameworks such as EMYCIN and EXPERT. However, it is well to remember that the general purpose language environments offer greater flexibility because, as noted, they are not tied to a particular control or inference strategy. The word 'shell' should be used to describe an expert system framework within which a committment has been made to a particular inference strategy. It follows, of course, that use of a particular shell requires a good match between the shell's control and representation strategies and the task to be captured.

CONSIDERATIONS IN CHOOSING A DEVELOPMENT ENVIRONMENT

Since the goal of an expert system project is the design of a prototype system that demonstrates the feasibility of capturing some task in an expert system, the implementation phase should in general be performed in an environment that allows a spectrum of representation and control schemes to be invoked and played with. Thus, from this point of view, one would tend to downplay issues of efficiency and ease of interfacing to any related analysis, modeling or simulation programs that might form part of the domain knowledge. However, there are still many things to think about in choosing the right environment (consisting of hardware, basic programming language, and shell or tool) for developing an expert system and, subsequently, for using it in a run-time environment. Some of the issues are explicated in this and the next sections.

For example, an expert system shell that embodies the fewest *a priori* constraints in building an application system on top of a core representation is probably most desirable. That is, it is best to operate on a principle of least committment to a single representation scheme or a single control approach until there is complete understanding of the issues of representation and inference. However, this is not a universally applicable principle because it assumes that each expert system project is a research project. In those instances where the representation and control issues are well understood from the beginning, then a more restrictive tool can be much easier to use. If the paradigms match, then a lot of the work that needs to be done is already incorporated into the tool.

Among completely general environments and representation languages, several choices are available [see (Harmon and King, 1985) for a relatively complete listing], although cost and hardware considerations may limit the use of such general tools. Intellicorp's KEE is similar to Xerox's LOOPS: both allow integration of frame and rule representations, coupled to several different control structures, together with features of procedural and access-oriented (also known as incorporating 'active values' or 'demons') programming. KnowledgeCraft, developed by the Carnegie Group, could be a good choice for some exploration and applications since it embodies frames, backward-chaining and forward-chaining, and also has hooks for system development.

Such general tools are run on LISP machines, sometimes (still) in different dialects, and sometimes the same product runs in different dialects on different machines. The preferred choice of LISP dialect is fairly clear, however, especially in light of the Common LISP standard. Thus, one would ordinarily select a different dialect of LISP only if the available implementation of Common LISP is inefficient. Of course this can be dealt with when moving to the run-time environment. Another reason for choosing an alternative dialect is that the software desired may not exist in Common LISP, or the machine available may not have an implementation of Common LISP—although this is not likely to happen in the future given the acceptance of the Common LISP standard.

The quality of the Common LISP implementation is an additional consideration in the joint decision of Common LISP and hardware processor. There are several metrics by which one might evaluate competing implementations, including memory requirements, the compactness of the code, its virtual memory requirements (locality), variable lookup and binding, data structure manipulation, type computations, arithmetic operations, and so on. Benchmarks for common LISP implementations have been proposed (Gabriel, 1985), but experience has not shown great uniformity or consistency in the application of these benchmarks to different Common LISP implementations when running on different platforms.

CONSIDERATIONS IN CHOOSING A RUN-TIME ENVIRONMENT

Run-time considerations are quite different from those outlined above and must perforce be addressed in detail after substantial progress has been made on

the shape of the prototype expert system, its coupling to and dependence on the development environment, and on the kinds of hardware and software that might be part of the delivered expert system. For example, the expert system might be designed as a pre- or post-processor to a numerical package, or it might be intended to work in a CAD system in a design environment. Thus, the particular nature of the intended run-time environment will influence strongly the way that the expert system is delivered.

However, some of the issues can be addressed here and, to give the discussion some concreteness, it will be cast into the context of considering an expert system as an intelligent pre- and post-processor for analysis packages based on the finite element method (FEM). The ideas stem from the early stages of a collaborative research effort on the feasibility of such an application (Dym and Fenves, 1986, Fenves, 1985). The most important issue in tying an expert system to an FEM package concerns the means by which expert system modules can be incorporated into present (and future) FEM products that are (will be) commercially available.

Most FEM codes are programmed in FORTRAN, while the expert system modules will be developed in an AI programming environment or within an expert system shell. One way to interface the two would be through file transfers. This would be a very weak coupling that would inhibit information transfer between the two systems.

Another possibility for developing efficient interfaces is the use of a common operating system (e.g. UNIX) and its attributes (e.g. UNIX 'pipes'). However, given that many FEM codes are run in VAX environments, with their VMS operating systems, it is not clear how a match of operating systems would be made, especially since UNIX is used often in AI/LISP machines.

It would be much more desirable to have both the FEM code and the expert system modules written in the same language. This would allow tighter coupling between the two, and would also make it easier for vendors to develop and support their products. However, in the short run this seems unlikely as it would not be cost-effective to rewrite the expert system modules in FORTRAN. It also seems unlikely that the FEM codes will be recast in LISP. One middle ground that appears viable is the recoding of FEM codes into C. This might be possible because there appear to be under development some cross-compilers that would allow LISP programs to be compiled into C or into 68000 assembly code. A variation on this option is the coding of the expert system into C as well, an idea which has received some attention in the literature and the commercial world of tool developers. However, as will be noted below, the advent of new chips and the inherent advantages of LISP (in terms of memory management, programming environments, etc.) still leave LISP as the language of choice for expert system development and application (Barber, 1987).

There are two extremely important computational issues that must be addressed in this phase of the project: speed and memory. The issues arise because Common LISP is a very large language that is also memory intensive. It is also computationally intensive and runs relatively slowly (several times slower than C, for example). Thus, the following questions arise: Is the CPU fast enough to run LISP at a practical speed? Also, is there enough addressable memory to handle both LISP and the application? (On its own, LISP requires 2.5 to 3 MB. A more typical minimum memory requirement is 4 MB, with 8 MB usually being specified for more serious applications.)

Further, in terms of memory, it is of critical importance that the run-time environment contain sufficient primary memory to avoid unnecessary page faults and thrashing. It was noted earlier that 8 megabytes of memory would usually be needed in a development machine. This requirement may be reduced as more compact versions of LISP are developed, but it must be understood that these more compact versions do not exist yet, nor is it clear when they will exist, if at all, nor how effective they will be.

In terms of the FEM application being discussed here, the issues appear as follows: Assuming that an FEM analysis assistant can be successfully built in functionally-specific modules, can one or more of the modules be run in real-time, while the user waits, or will they be run off-line in batch mode, or even overnight? The answer to this question will clearly affect the nature of the user interaction and the manner in which the design process takes place. However, an answer to this question will not be available until after at least one module is developed, and even then there will be variations in the size and complexity of future modules.

In planning both the development and the run-time phases of a project such as that envisioned here, some things can be done to make the expert system more efficient. The first is that the LISP program should be compiled, although a LISP interpreter will still be required. Second, one can structure the rule-sets so that the user can control the order of application of the rules, thus avoiding unnecessary pattern-matching. Third, applying the notion of 'meta-level' control, one can partition the rules to ensure that only the immediately relevant rules are fired in any given computation.

Lastly, one can look for hardware that makes use of

the newer and faster LISP chips that are under development. Such chips are 36 bit chips, rather than 32, and they facilitate the tagged-memory architecture of LISP code by allowing type-checking, branching and stacking to be done concurrently in micro-code. These chips are expensive, but they are some five to ten times faster than current LISP chips and are expected to be available in the next year or two. In addition, with the advent of new processors such as the Intel 80286 and 80386, which are 16 and 32 bit chips, respectively, there is sufficient execution speed (1–2 mips for the 80286 and 3–4 mips for the 80386) and memory (up to 16 MB physical and 4 gigabytes virtual for the 80286 and 4 gigabytes physical and 64 terabytes virtual for the 80386) to support substantial expert system development and application (Barber, 1987).

It might be noted that, the above specifics notwithstanding, the world of machines, speed and memory is in a state of very rapid flux. Thus, the best advice may be to proceed only after absorbing the most recent information and anticipating that things will change rapidly in the future. Thus, to the extent one can favor open architectures and upward compatibility, in the most general terms, the better off one is likely to be.

5. Institutional issues

Institutional, or organizational aspects of expert system building are not only important in their own right, because of the commitment of resources required for the building of a worthwhile and robust system. They are also important because organizational issues interact strongly with some of the technical issues, often to the point where they cannot be considered as independent of each other. The key institutional issue, of course, is the allocation of resources, both to the system building project as a whole and among the various phases as the project proceeds. The impact of resource decisions will be felt on the choice and characterization of the task to be modelled, on the choice of the development environment, on the knowledge acquisition process, and on the delivery platform. By way of spotlighting the interaction of technical and institutional issues, it is worth looking at some examples of expert system projects that are considered as success stories.

The Dipmeter Advisor is an expert system designed to aid in the geological analysis of subsurface formations by examining and interpreting magnetic data taken from boring logs (Smith, 1984). Boring-log analysts look at output from strip-chart recorders as they try to interpret the geological data, and the Dipmeter Advisor maintains this ability, incorporating also various kinds of summary data and the ability to scroll smoothly and quickly among various strips of data. This required a large investment in the user interface for this expert system. In fact, the distribution of lines of code in the Dipmeter Advisor is as follows:

inference engine	8%,
knowledge base	22%,
feature detection	13%,
user interface	42%,
support environment	15%.

Note that 42% of the code is taken up by the user interface, while the amount devoted to the obvious expert system components, the knowledge base and the inference engine, is only 30% of Dipmeter Advisor's code. This emphasis on the user interface is not usual in expert system design. In the PRIDE system, for example, about 40% of the code is dedicated to the user interface (Mittal *et al.*, 1986).

This is an interesting example of how technical and institutional implementation issues interact because it speaks directly to the issue of how resources are allocated as an expert system is being built. It is important to recognize that the software created (and the delivery platform used) must create an environment consistent with how the expert exercises his expertise. Thus the knowledge engineers must put forward an interface that can and will be used by the experts and other potential users. Again, in the PRIDE system, the starting point in the design process captured there is an interface that allows the designer/user to 'sketch' a path for a paper handling subsystem in a way that is very close in feeling to how it would have previously been done on a drawing board (Mittal *et al.*, 1986, Morjaria *et al.*, 1985).

Dipmeter Advisor is also an interesting example because of the time taken for its development and evolution. It began as a research project in 1978, and the first prototype was completed in 1980. The prototype, containing 245 K bytes of DEC 2020 LISP code and another 450 K bytes of VAX FORTRAN code, was too slow. The second implementation was completed in 1983, running on a Xerox workstation with 650 K bytes of Interlisp-D code. The 1983 implementation was considered sufficiently fast and robust for testing (Smith, 1984).

Another well-known expert system also started as a research project in 1978 (McDermott, 1981). The R1/XCON system began at Carnegie-Mellon University in that year, with the first prototype being

delivered to DEC in 1980. However, it was not put into regular use there until 1982, and even now it requires a very large support and maintenance staff. (Some fifty people work on this and related projects at DEC.) However, it does successfully configure 97% of all VAX orders, and thus its economic return is quite substantial.

Both Dipmeter Advisor and R1/XCON are often cited to make the point that building a large and robust expert system is a long and expensive process. Both of these projects required years of research and experimentation before they were adopted for regular use. However, both of these systems were started long before the availability and understanding of the kinds of tools that are now taken for granted. It is not that building large, 'industrial-strength' expert systems has become cheap, but the process is not as expensive or as time consuming as the Dipmeter Advisor and R1/XCON projects would indicate. Both the PRIDE system and the LSC Advisor were brought to working prototype stage within 12–18 months, and the resource consumption of both these design system projects was far less than the other projects just cited (Dym *et al.*, 1987, Mittal *et al.*, 1986).

One of the less obvious costs in building such systems is that of the experts' time. Clearly, while the experts are involved in a system building project they will be doing less of what they would otherwise be doing in their various organizations. Further, without the expert's active, enthusiastic and continuing interaction with the knowledge engineers and other system builders, the knowledge cannot be successfully explicated and captured in the system. Thus, major commitments must be made by the experts and by their organizations. Without such commitments, which obviously can be very expensive, serious system building endeavors should not be undertaken.

Another aspect of expert system building that requires major institutional commitments is the life of the system after its development. An expert system is—or at least should be—a dynamic system that needs active maintenance and updating. New cases and situations provide further experience that can be added to the knowledge base, as can refinements of the user interface and other features of the system. (Recall that one of the real advantages of the separation of knowledge and control in expert system programming is that it facilitates the addition of 'chunks' of knowledge (Dixon and Dym, 1986, Dym, 1985*a*). This requires a commitment by management to maintain and support the system, which really means that the system should provide a continuing benefit sufficient to justify to its sponsoring organization this continuing expense.

6. Speculations on future system implementations

This section is devoted to a somewhat specialized discussion of future expert system possibilities. The motivation comes from extrapolation from a current research project (Dym *et al.*, 1987), taking account also of current research in distributed processing at the University of Massachusetts (Durfee *et al.*, 1985).

The current project involves the development of an expert system to do architectural code checking (Dym *et al.*, 1987). It is clear from architectural and engineering practice that building and other structural designs often need to meet more than one set of code requirements. Indeed, the interaction of multiple constraints sets in a design project is only one example of how expertise in engineering projects may be distributed, making the interaction of multiple sources of knowledge a serious engineering issue. Other examples include the interaction of subsystem designers as an overall system design is assembled (out of the various subsystems), the interaction of multiple experts in achieving a diagnosis or in developing an analysis strategy for a complex problem, and the formulation of a coherent set of plans for a project from the individual plans of several involved parties.

It might be noted that the distribution of expert systems and their interaction is not a dream for the distant future. It is already clear that some of the benefits of building (single) expert systems derive from the ability to use them at a variety of sites. Such use facilitates standardization of techniques, methods and requirements among geographically dispersed parts of an organization. It also allows localized expertise to be distributed for training purposes, and it allows the development of a community memory for an institution (Dym, 1985*a*, Mittal and Dym, 1986). The point is that the benefits of distributing a single expert system are so clear and obvious that the perceived possibilities of concurrent expert systems will surely drive innovation in this area, especially with the increasingly rapid development of parallel processor hardware.

The study of the representation of and solutions to these kinds of problems comprises that branch of artificial intelligence usually called 'distributed AI'. And perhaps the major issue is that of coupling between the knowledge sources or expert systems. The ideal scenario, of course, is that of *tight coupling*, where operating systems and memory are shared, and there are no synchronization problems. This is an ideal, however, that cannot typically be met in a complex engineering environment in which there is already established a web of computers, peripherals,

and the software and other hardware necessary to achieve the objectives of a particular organization.

As noted above, a more typical environment in which expert systems will interact is one wherein they are *loosely coupled*. This denotes the situation wherein each system could have its own data input and will communicate with other expert systems over a low-bandwidth channel. Clearly, in order to think about the interactions of concurrently executing expert systems, it has to be accepted that the individual systems should not be designed as isolated, closed expert systems. Rather, the individual expert systems must be open and capable of a high degree of interaction. Some of the research questions to be faced in the development of complexes of expert systems are as follows:

(1) How does an expert system solve subproblems that interact?

(2) How is parallel processing done over a set of expert systems?

(3) What is the appropriate representation scheme for the highly abstract and processed data that will be shared by the cooperating systems?

(4) What is the control mechanism that governs the set of cooperating systems, and how does it interact with the individual control systems of each knowledge-based system?

These are only some of the research questions to be addressed for an understanding of how cooperating knowledge sources can be used to perform better engineering. The kind of architecture typically proposed for an ensemble of cooperating systems is called the *blackboard* (Nii, 1986*a*, 1986*b*), and a specific example of such an architecture is the *Generic Blackboard* (GBB) architecture developed at the University of Massachusetts (Corkill *et al.*, 1986). But the main point is that, new as they are, expert systems are rapidly evolving, and as they do so they are becoming more open and thus more widely applicable to important engineering applications.

7. Conclusions

This article has been devoted to a discussion of the implementation of an expert system, that is, of how to make it happen. In the process many important issues have been raised, some technical, some institutional, some interactive between these two dimensions. It is probably useful to close this discussion with some warnings or maxims about expert system development, as well as with a listing of some of the advantages to an organization that sponsors the development of an expert system.

As maxims, it is worth remembering that:

* Expert systems cannot do the impossible, e.g., cure cancer.
* Expert systems cannot do the extraordinary, e.g., make money on the stock market.
* Knowledge is expensive.
* It takes time to build a serious, robust expert system.
* A single knowledge representation scheme is often inadequate.
* Much more is known about developing diagnostic and interpretive expert systems than about planning and design systems.

Then why invest in an expert system? Because:

* Expert systems do not get tired. They can perform routine tasks with high reliability and consistency.
* The knowledge acquisition process deepens and sharpens the experts' own understanding.
* Expert systems allow the experts to concentrate on rarer, more interesting tasks.
* Expert systems can be used to train neophytes.
* Expert systems provide a community memory for sharing and propagating knowledge.
* Expert systems, with networking, permit the widespread standardization of techniques, methods, requirements, etc.

Acknowledgements

The author is grateful for helpful comments from several collaborators: S. Gonick of Amerinex Artificial Intelligence, Inc., and D. D. Corkill, R. P. Henchey and E. A. Delis of the University of Massachusetts. E. M. Riseman of the University of Massachusetts provided useful comments on an early version of Section 4 of this paper.

References

Barber, G. R 1987 LISP vs. C for implementing expert systems, *AI Expert* **2**(1), 28–31.

Bobrow, D. G., Mittal, S. and Stefik, M. J. 1986. Expert systems: Perils and promise, *Communications of the ACM* **29**(9), 880–894.

Buchanan, B. G. *et al.* 1983. Constructing an expert system. In: Hayes-Roth, F., Waterman, D. A. and Lenat, D. B., Eds, *Building Expert Systems*, Reading, MA: Addison–Wesley.

Corkill, D. D., Gallagher, K. Q. and Murray, K. E. 1986. GBB: A generic blackboard development system. In: *Proceedings of the Fifth National Conference on Artificial Intelligence, Vol. II*, Philadelphia, PA, pp. 1008–1014.

Dixon, J. R. and Dym, C. L. 1986. Artificial intelligence and geometric reasoning in manufacturing technology, *Applied Mechanics Reviews* **39**(9), 1325–1330.

Durfee, E. H., Lesser, V. R and Corkill, D. D. 1985. *Coherent Cooperation Among Communicating Problem Solvers,* Technical Report 85-15.Amherst, MA: Department of Computer and Information Science, University of Massachusetts.

Dym, C. L. 1985*a*. Expert systems: New tools for computer-aided engineering, *Engineering with Computers* **1**(1), 9–25.

Dym, C. L. (Ed.) 1985*b*. *Applications of Knowledge-Based Systems to Engineering Analysis and Design* New York: American Society of Mechanical Engineers.

Dym, C. L. and Fenves, S. J. 1986. Feasibility study of a knowledge-based expert system finite element modeling and analysis assistant (FEMAA), a joint research project at the University of Massachusetts and Carnegie–Mellon University sponsored by the U.S. Air Force Office of Scientific Research, 1986–1987.

Dym, C. L., Delis, E. A. and Henchey, R. P. 1987. Representation and control issues in automated architectural code checking, manuscript in preparation

Fenves, S. J. 1985. A framework for a knowledge-based finite element analysis assistant. In: C. L. Dym (Ed.), *Applications of Knowledge-Based Systems to Engineering Analysis and Design,* New York: American Society of Mechanical Engineers.

Gabriel, R. P. 1985. *Performance and Evaluation of Lisp Systems,* MIT Press, Cambridge, MA, 1985.

Harmon, P. and King, D. 1985. *Expert Systems,* New York: John Wiley.

Hayes-Roth, F., Waterman, D. A. and Lenat, D. B. 1983. (Eds), *Building Expert Systems,* Reading, MA: Addison-Wesley.

Maher, M. L., Sriram, D. and Fenves, S. J. 1984. *Tools and Techniques for Knowledge-Based Expert Systems for Engineering Design,* Technical Report DRC-12-22-84. Pittsburgh, PA: Design Research Center, Carnegie–Mellon University.

McDermott, J. 1981. R1: The formative years, *AI Magazine* **2**(2), 21–29.

Mittal, S. and Dym, C. L., 1985. Knowledge acquisition from multiple experts, *AI Magazine* **6**(2), 32–36.

Mittal, S., Dym, C. L. and Morjaria, M. 1986. PRIDE: An expert system for the design of paper handling system, *Computer* **19**(7), 102–114.

Morjaria, M., Mittal, S. and Dym, C. L. 1985. Interactive graphics in expert systems for engineering applications. In: *Proceedings of the 1985 International Computers in Engineering Conference and Exhibit,* Boston, MA, pp. 235–241.

Mullarkey, P. W. 1987. Languages and tools for building expert systems. In: Maher, M. L., Ed., *Expert Systems for Civil Engineering: Technology and Applications,* New York: American Society of Civil Engineers.

Nii, H. P. 1986*a*. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures, *AI Magazine* **7**(2), 38–53.

Nii, H. P. 1986*b*. Blackboard systems; Blackboard application systems, blackboard systems from a knowledge engineering perspective, *AI Magazine* **7**(3), 82–106.

Prerau, D. S. 1985. Selection of an appropriate domain, *AI Magazine* **6**(2), 26–30.

Rehak, D. R. and Fenves, S. J. 1985. Expert systems in civil engineering, construction and construction robotics, *1984 Annual Research Review,* Pittsburgh, PA: Robotics Institute, Carnegie–Mellon University.

Sheil, B. 1983. Power tools for programmers, *Datamation* **29**(2), 131–144.

Smith, R. 1984. On the development of commercial expert systems, *AI Magazine* **5**(3), 21–34.

Waterman, D. A. 1985. *A guide to expert systems,* Reading, MA: Addison-Wesley.

Clive L Dym is Professor of Civil Engineering and Adjunct Professor of Computer and Information Sciences at the University of Massachusetts, Amherst, where he has also been Head of the Department of Civil Engineering (1977–1985). He was a Senior Scientist at Bolt Beranek and Newman in Cambridge MA (1974–1977), and served on the faculties of SUNY Buffalo (1966–1969) and Carnegie–Mellon University (1970)–1974). He has held visiting appointments at the Technion–Israel Institute of Technology (1971), the Institute for Sound and Vibration Research at the University of Southampton (1973), and Stanford and the Xerox Palo Alto Research Center (1983–84). Dr. Dym completed undergraduate work at the Cooper Union (1962), received an MS from the Polytechnic Institute of Brooklyn (1964) and a PhD from Stanford University (1967). Dr Dym has done research on a variety of problems in applied mechanics and acoustics. Recently his research activities have focused on the development of expert (knowledge-based) systems for engineering analysis and design. Dr Dym's research results have been published in some 60 journal articles and in seven books. He was the recipient of the 1980 Walter L Huber Research Prize of the ASCE and the Western Electric Fund Award of the New England Section of the ASEE (1983). He is on the Editorial Board of the Journal of Sound and Vibration, has been an Associate Editor of the Journal of the Acoustical Society of America, and is founding Editor of the new journal, Artificial Intelligence for Engineering Design, Analysis and Manufacturing, published by Academic Press.