16

XSEL: a computer sales person's assistant

J. McDermott
Carnegie-Mellon University
Pittsburgh, USA

Abstract

R1, a knowledge-based configurer of VAX-11 computer systems, began to be used over a year ago by Digital Equipment Corporation's manufacturing organization. The success of this program and the existence at DEC of a newly formed group capable of supporting knowledge-based programs has led other groups at DEC to support the development of programs that can be used in conjunction with R1. This paper describes XSEL, a program being developed at Carnegie-Mellon University that will assist salespeople in tailoring computer systems to fit the needs of customers. XSEL will have two kinds of expertise: it will know how to select hardware and software components that fulfil the requirements of particular sets of applications, and it will know how to provide satisfying explanations in the computer system sales domain.

Introduction

The world is filled with tasks that can be performed satisfactory only by those who have acquired, through apprenticeship, the bodies of knowledge relevant to these tasks. In designing programs to perform these tasks, the AI community has had to face the issue of how to represent large amounts of ill-structured knowledge in a way that enables relevant knowledge to be quickly brought to bear. The software tools that have been developed for this purpose are quite different from the tools developed for use in well-structured domains; in particular, they are intended to be used to implement continuously evolving programs. Since only a handful of people are familiar with any of these tools, and since the programs they can be used to implement acquire expertise only with use, the tools are currently under-utilized. One approach to this technology transfer problem is for people familiar with the tools to develop programs that have some expertise in various industrial domains, plant the programs in those domains, and let those who need the programs acquire familiarity with the tools by using (and continuing to develop) the programs.

This is essentially the strategy that Carnegie-Mellon University and DEC adopted, and it appears to be working well. In December 1978, work was begun at CMU on R1 (McDermott 1980a, 1980b) a program that configures VAX-11 computer systems. In less than a year, R1 had been developed to a point where it could be used on a regular basis by DEC's manufacturing organization to configure VAX-11/780 systems. At that time, DEC established a group responsible for the maintenance and continued development of R1. During 1980, the group grew to include 12 people; it now consists of a manager, 5 people proficient in OPS5 (the production system language in which R1 is written), 3 people responsible for ensuring that R1's database (a set of descriptions of hardware and software components) is accurate and up to date, and 3 people responsible for developing processes to facilitate R1's use and to aid in extracting configuration knowledge from experts. Over the course of the year, CMU provided considerable technical assistance to this group. The group is now essentially self-sufficient; that is, it needs no assistance from CMU in maintaining and extending R1. In the Fall of 1980, the group implemented a version of R1 that can configure VAX-11/750 systems; almost all VAX systems built within the United States are now configured by these programs.

Though DEC now has the expertise required to continue the development of R1 and plans to extend R1's capabilities so that it will be able to configure PDP-11 systems, it is not yet quite at the point where it could embark on the development of knowledge-based programs that perform quite different tasks. Yet there is a need at DEC for such programs. In particular, R1 performs only the component organization part of what could be called the computer system configuration design task. The configuration design task has two parts:

- Component selection: the set of components that are both necessary and sufficient to satisfy the requirements imposed by a set of applications must be selected.
- Component organization: the set of components selected must be organized to form a system that satisfies the requirements imposed by the set of applications.

A program that could perform the component selection part of the task would be extremely valuable. Thus DEC has asked CMU to develop such a program.

In the next section, R1's capabilities will be briefly reviewed. Then in section 2, XSEL, the program being developed to perform the component selection part of the configuration design task, will be described. Though XSEL is still in the initial stages of development, it has become apparent that the recognition-driven approach to problem-solving that characterizes R1 can also be the principal approach used by XSEL. The fact that two tasks of a significantly different nature are both amenable to a recognition-driven approach suggests that there may be a wide range of tasks for which this approach is appropriate. The final section will discuss XSEL's explanation capability. Because some of the people with whom XSEL will interact will have only a limited understanding of con-

figuration design issues, it is important that XSEL be able to answer a wide range of questions in a simple and straightforward fashion.

1. R1's ROLE IN CONFIGURATION DESIGN

R1 takes a set of components as input and produces diagrams showing what the spatial relationships among the components should be. Though it knows almost nothing about the subtask of selecting a set of components to satisfy a functional specification, it does understand that certain components, in order to be configured, may require other components. If the set of components it is given is incomplete in this sense, it adds whatever components are required to make the set configurable. R1 has a relatively large amount of knowledge that enables it to recognize the acceptable ways in which components can be associated under various conditions. It uses this knowledge to construct a single configuration that satisfies all of the organizational constraints. Because its knowledge is sufficient to enable it to recognize what to do next at each step, it performs this task with almost no search; that is, it seldom needs to backtrack.

R1 is implemented in OPS5, a general-purpose, rule-based language (Forgy 1980, Forgy 1977) OPS5 provides a rule memory, a global working memory, and an interpreter that tests the rules to determine which ones are satisfied by a set of the descriptions in working memory. A rule is an IF-THEN statement consisting of a set of conditions (patterns that can be matched by the descriptions in working memory) and a set of actions that modify working memory. On each cycle, the interpreter selects one of the satisfied rules and applies it. Since applying a rule results in changes to working memory, different subsets of rules are satisifed on successive cycles. OPS5 does not impose any organization on rule memory; all rules are evaluated on every cycle. It often turns out to be convenient, however, to be able to restrict the set of rules that can fire on any given cycle to those that bear on the task currently being performed; this can be accomplished by including in each rule a condition element that specifies the context (subtask) in which the rule is relevant.

In implementing R1, OPS5's two memories were augmented with a third. This memory, the data base, contains descriptions of each of the more than 750 components currently supported for the VAX-11. Each database entry consists of the name of a component and a set of 18 or so attribute/value pairs that indicate the properties of the component which are relevant for the configuration task. As R1 begins to configure an order, it retrieves the relevant component descriptions. As the configuration is generated, working memory grows to contain descriptions of partial configurations, results of various computations, and context symbols that identify the current subtask.

Production memory contains all of R1's knowledge of how to configure VAX-11 systems. R1 currently has about 850 rules that enable it to perform the task. These rules can be viewed as state transitions operators. The conditional part of each rule describes features that a state must possess in order for the rule to be applied. The action part of the rule indicates what features of the

state have to be modified or what features have to be added in order for a new state that is on the solution path to be generated. Each rule is a more or less autonomous piece of knowledge that watches for a state that it recognizes to be generated. Whenever that happens it can effect a state transition. If all goes well, this new state will, in turn, be recognized by one or more rules; one of these rules will effect a state transition, and so on until the system is configured. An English translation of a sample rule is shown in Fig. 1.

```
ASSIGN-UB-MODULES-EXCEPT-THOSE-CONNECTING-TO-PANELS-4

    IF: THE MOST CURRENT ACTIVE CONTEXT IS ASSIGNING DEVICES TO UNIBUS MODULES
        AND THERE IS AN UNASSIGNED DUAL PORT DISK DRIVE
        AND THE TYPE OF CONTROLLER IT REQUIRES IS KNOWN
        AND THERE ARE TWO SUCH CONTROLLERS
                NEITHER OF WHICH HAS ANY DEVICES ASSIGNED TO IT
        AND THE NUMBER OF DEVICES THAT THESE CONTROLLERS CAN SUPPORT IS KNOWN
  THEN: ASSIGN THE DISK DRIVE TO EACH OF THE CONTROLLERS
        AND NOTE THAT THE TWO CONTROLLERS HAVE BEEN ASSOCIATED
                AND THAT EACH SUPPORTS ONE DEVICE
```

Fig. 1 – A sample rule.

Though R1 performs its task adequately, it is a less valuable tool than it might be because it lacks certain pieces of information. The only information that is currently made available to R1 is the set of components ordered. Since R1 has no knowledge of the physical characteristics of the room or rooms in which the system is to be housed, it cannot produce a realistic floor-layout even though it has the capability to do so; this means that it cannot determine the precise lengths of cable required to connect some pairs of components. Moreover, R1 has no access to information that would enable it to determine whether the intended uses of a system imply unusual configuration constraints; thus the configuration that R1 produces is not necessarily the one that provides the customer with the best performance for his particular set of applications.†

2. XSEL's ROLE IN CONFIGURATION DESIGN

XSEL's role complements R1's. XSEL's task is to select the set of components that satisfy the requirements imposed by a set of applications. It then informs R1 of its selections and provides any additional information R1 will need in order to tailor its configuration to those applications. It is important to note that one of R1's capabilities makes XSEL's task considerably easier than it would otherwise be. Much of the problem of component selection is that salespeople, in addition to having to select components directly relevant to the intended uses of a system, also have to be concerned with 'support' components (e.g., back-

† The most recent extension to R1 is a capability that enables it to accept as part of its input a set of *ad hoc* (customer-specific) constraints; in producing a configuration, R1 gives preference to these constraints over the ordinary-case constraints encoded in its rules (McDermott 1981).

planes, boxes, panels, cabinets, cables, etc). Since part of R1's task is to make sure that all such support components are included on the order and to add them if they are not, there is no need for XSEL to concern itself with support components at all. Thus the component selection problem reduces to the problem of selecting just that set of components that would normally be included in a system's functional specification.

2.1. The humble role

XSEL in its current state is little more than a front end for R1. It allows a user to specify a cpu, some amount of primary memory, whatever software is desired, and whatever devices are desired (e.g., disk drives, tape drives, terminals, printers, etc.); this skeletal order is then passed to R1 to be fleshed out and configured. The interaction with the user actually has three stages:

(1) The user is asked a few standard questions: his name, the order identification number, etc.
(2) The user is asked what components he wants to order.
(3) The user is asked for information required by R1 in order to do floor-layout and is asked to indicate any special configuration constraints that the intended uses of the system imply.

The first stage is simple and straightforward; the second and third stages are somewhat more interesting.

When asked what components he wants to order, the user may, if he wishes, enter the names (and quantities) of the components he wants; XSEL then performs a few simple consistency checks to make sure that the components ordered are compatible and goes on to the third phase. Since DEC's naming conventions are somewhat confusing, even to the initiated, the user may specify the components he wants by 'subtype' rather than by name. DEC's names for components all have the form xxxxx-yy; the first five characters indicate the subtype and the final two characters indicate the variation. Typically a subtype will have anywhere from four to eight variations. Partly as a matter of convenience and partly to avoid the errors that frequently crop up because an incorrect variation has been specified, the user may elect to specify some or all of the components by subtype. If so, XSEL will ask a few questions to determine which variation the user wants.

The user may specify some types of components in terms of total capability desired, rather than by name or subtype. Currently, primary and secondary memory can be ordered by specifying megabytes desired, and printers can be ordered by specifying total printing capability desired in lines per minute. In order to handle this type of specification, XSEL does need some knowledge of how to select among subtypes. Since specification in terms of capability does not ordinarily narrow the set of possibilities to the variations of just one subtype, XSEL must have criteria that enable it to make reasonable choices. Currently XSEL has a few rules that enable it to avoid obviously poor choices; it will

need a significant amount of additional knowledge before its choices will be consistently adequate.

Once the user has specified the components he wants, XSEL goes on to the third stage. Here, XSEL asks the user a number of questions about the room or rooms in which his system will be housed. It asks for the dimensions of the rooms and the positions of doorways and obstructions. If the user wishes, he may then specify how he wants some or all of the components on the order to be positioned; he may specify the precise location of components or may indicate the approximate locations of various groups of components. After the user has provided as much floor layout information as he wants, he is given the opportunity to enter other configuration information. He can specify how devices are to be distributed among controllers, the type of length of cable to be used to connect a pair of devices, the positions of controllers on buses, and the positions of backplanes in boxes or of boxes and panels in cabinets. XSEL then passes to R1 the set of components ordered and any other information the user entered.

Much of the knowledge that the initial version of XSEL requires is not domain-specific, but is rather knowledge of how to lead a user through a selection process. And the relatively small amount of configuration design knowledge required is primarily knowledge of the task's structure. Given the penchant of many knowledge engineers for building special-purpose 'engines' (ordinarily in LISP) to contain knowledge of this sort, it is worth indicating why all of XSEL's knowledge is represented in the form of rules. There are several reasons:

- It is important that a user be able to take as much or as little advantage of XSEL's expertise as he wants at any point during an interaction; we suspect that this flexibility will be easier to achieve if all of XSEL's knowledge is uniformly represented.
- The domain knowledge that XSEL currently has provides a base that will support more specific knowledge of how to perform the component selection task; it is important that this base knowledge be neither more nor less privileged than the knowledge that will be added.
- In order to construct explanations that focus attention on the most significant steps in its decision-making process, a program needs to be able to examine that process.

In the remainder of the paper, the strategy for developing XSEL's expertise in the domains of configuration design and computer system sales explanation will be discussed. Although the knowledge that XSEL currently has in these domains is extremely limited, the structure needed to support a large amount of much more specific knowledge appears to be in place.

2.2 A more exalted role

A considerable amount of knowledge is required for the configuration design task because there is no small set of general principles that can be used as the basis of component selection. A customer is often not completely sure of the

uses to which he will put his computer system, and even when he is, may have little idea of how much functionality each use requires. The best he can do is provide indirect measures of the requirements of each of his intended applications. Thus an expert must know what sorts of data to collect and what can be inferred from that data. Furthermore, each intended use is independent from the point of view of functionality required; that is, knowing how to infer the resources required for one application does not get one very far in inferring the resources required for a different application. Finally, there are ordinarily a number of different combinations of components that supply essentially the same functionality; the differences among these sets of components are usually quite small, but some may be better suited to the particular needs of the customer than others.

Though a significant amount of knowledge is required in order to perform the component selection task adequately, all of this knowledge is relevant to one of the following three subtasks:

- Decide whether a particular component type is necessary for one or more of the intended applications.
- If some type of component is needed, select from among the subtypes available that subtype which best fulfils the need.
- If a subtype has been selected, select from among the variations available that variation which best fulfils the need.

The most striking characteristic of these subtasks is that for all three the decisions to be made are relatively independent both of prior and of subsequent decisions; thus the tasks are fundamentally recognition tasks. The task of determining whether a particular type of component is necessary for some application involves little more than determining whether the conditions that signal the need for that component type are satisfied by the application. The task of selecting a particular subtype or variation involves little more than determining which of the competing sets of conditions that signal the need for the possible alternatives are satisfied by the application. It is of course true that in order for the component selection task to be performed by recognition, all of the relevant information about each intended application must be known. But this is just another recognition task – the task of recognizing what questions must be asked before a selection can be made.

Though there are a few complicating factors that will be discussed below, the fact that the component selection task can be recognition driven makes XSEL conceptually quite simple. Given a user who wants assistance in selecting a set of components, XSEL first finds out what general classes of applications he has in mind. Each of these applications will suggest some number of component types as possibly necessary. Given a set of applications and a possibly necessary component type, a set of questions will suggest themselves; the user's answers to these questions will enable XSEL to recognize whether or not a component of that type is necessary. Once a component type is determined to be necessary,

additional questions will suggest themselves; the user's answers to these questions will enable XSEL to recognize which subtype (and then which variation) best satisfies the requirements imposed by the applications. Though XSEL's knowledge of how to do component selection is currently quite limited, it should be clear that adding knowledge presents few problems. Each of XSEL's component selection rules recognizes a particular need in the context of a particular application; rules are almost completely independent of one another, and thus modifying or adding rules has no hidden implications.

How difficult it is to recognize that a particular component type is required for a particular application depends on the nature of the application. When an application always requires the functionality provided by some particular type of component, all that is needed to ensure that that type of component will be ordered is a rule that associates the application with the component type. If the application requires that type of component only under certain conditions, then several rules may be necessary, each of which recognizes one set of conditions. When several applications all require the same component type, it is necessary to distinguish cases in which the component can be shared from those in which a separate component is required for each application.

Before a subtype of a required component type can be selected, the total capacity that must be provided by the component type must be known. This is trivial when the capacity of a component type is fixed (e.g., a FORTRAN compiler, a floating point accelerator). But in cases of component types with variable capacity, XSEL must perform a computation. Each of its rules that associates an application with a variable capacity component type also represents the form of the required computation and specifies what information has to be extracted from the user in order for the computation to be performed. This knowledge is represented as a set of elements that collectively define the computation. Each of these elements describes the operation that must be performed to find the value of one of the terms in the computation; for the primitive terms, the operation is to ask the user. Once the user supplies the information requested of him, the total capacity needed is computed. Since many of XSEL's computations rely on some pieces of information from the user, it is important that the computations be able to share information. Thus, if sometime during his interaction with XSEL, the user has supplied a value for a term that reappears in a subsequent computation, the later occurrence of the term inherits its value from the first occurrence.

An additional complication arises when the total capacity required depends on several different applications. It is ordinarily necessary in this case to do more than simply accumulate the values returned by the various application-specific computations. A variety of computations for each application must often be performed, and then the results of these computations must be combined. For example, to determine the amount of primary memory needed on a system, each application must compute the core requirements both of its compute-bound and its i/o bound jobs; the total amount of core required is a function of

the sum of core requirements of the compute-bound jobs and the maximum of the core requirements of the i/o bound jobs.

Since most component types have a number of subtypes, for XSEL to decide which of those subtypes to select (and in what quantity), it must ordinarily have more information than just a measure of the total capacity required. There are three sources for this information. Sometimes XSEL must ask the user for additional information about the intended uses of his system. Sometimes the information collected to determine total capacity implicitly contains the necessary information. And sometimes XSEL falls back on domain-specific heuristics to discriminate among candidates. When XSEL has collected as much information as it can, it searches its database of component descriptions for components that satisfy all of the constraints implied by this information. If all of the components found have the same subtype, then the subtype selection task is finished. If components of more than one subtype are found, XSEL must decide which subtype or subtypes to select on the basis of more general heuristics.†

Once a subtype is selected, XSEL must decide which variation is most appropriate. As we have seen, the initial version of XSEL asks the user a set of questions sufficient to discriminate among the possibilities; but these questions assume that the user knows which variations are best suited to his applications. If the user is relying on XSEL for guidance in selecting an appropriate set of components, it is quite likely that he will not know how to answer the questions, In this case, XSEL must use less direct means to collect the necessary information. The available sources are essentially the same as those used in obtaining the information needed to discriminate among subtypes. XSEL can ask for additional information about the intended uses of the system, make inferences from the information already collected, or fall back on domain-specific heuristics. Given a subtype and information further specifying that subtype, XSEL retrieves a component of that subtype from its database and adds the component to the order.

While engaged in the task of selecting an appropriate set of components, XSEL watches for indications that the intended use of some subset of components implies unusual configuration constraints. It has a set of rules that recognize situations that signal atypical use. If such a signal is present, XSEL generates a constraint that will cause R1 to tailor its configuration to fit the atypical use. Since XSEL communications with R1 by means of a simple language consisting a relatively small number of command forms, these constraints are easy for XSEL to generate.

3. THE EXPLANATION TASK

Several of the knowledge-based programs developed over the past few years have quite sophisticated explanantion capabilities. But for the most part, these programs do not treat explanation as a task that requires intelligence. Almost all

† Currently if XSEL must decide among subtypes of apparently equal appropriateness, it selects the least costly subtype.

of the knowledge these programs have is knowledge of how to solve problems in their respective domains. Little if any of it is knowledge of how to effectively communicate an understanding of the problem solving process to a user.† A major component of XSEL's knowledge will be knowledge of how to construct explanations in the computer system sales domain so that they contain just that information in which the user is interested.

XSEL currently has a few limited explanantion capabilities. It can provide the following kinds of information on demand:

(1) Descriptions of components, definitions of properties of components, definitions of terms used in formulas.
(2) Values of the properties of components, data entered by the user and inferences drawn from that data, components selected, configuration constraints generated.
(3) Significant differences among components of the same subtype, significant differences among subtypes of the same type.
(4) Its reasons for selecting particular components, its reasons for generating a configuration constraint.
(5) A justification of the reasoning that led it to select some quantity of components.

These capabilities vary greatly in the amount of knowledge about explanation that they presuppose. Capabilities 4 and 5 require the most knowledge and thus are the least developed at the moment; this section will focus primarily on these two capabilities and indicate what plans we have for developing them.

It should be noted that XSEL can provide any of the various sorts of information at any time during an interaction. As mentioned above, XSEL's rules are grouped into contexts on the basis of the tasks for which they are relevant. One of XSEL's contexts contains rules that determine, on the basis of how the user responds, whether he is entering a value that XSEL requested or wants something explained. The user indicates that he wants something explained by entering one of five types of commands. If the user gives one of these commands rather than entering the requested value, a rule that recognizes that command fires and generates a working memory element indicating the name of the context that can provide the sort of information he wants. This context contains all of the rules relevant to that sort of explanation.

The first three capabilities listed above are essentially information retrieval capabilities; XSEL needs only a small amount of general knowledge in order to provide the information desired. XSEL has access to a database of definitions; thus the first capability is achieved simply by look-up. The second capability is also achieved by look-up, though here there are two sources that might contain the information. If the value requested is the value of a property of a component, XSEL retrieves the value from its database of component descriptions;

† A notable exception is GUIDON (Clancey 79), a rule-based, tutorial program that can be built on top of MYCIN-like expert systems.

if the information is contained in a working memory element (e.g., the value of some term in a computation), XSEL responds with the value contained in that working memory element. The third capability allows somewhat more room for various forms of assistance, but is currently implemented simply. Since the differences among variations of the same subtype and among subtypes of the same type are ordinarily differences in just a few properties, XSEL searches through its database of component descriptions for the relevant set of components and then displays for each component those of its properties which distinguish it from at least one of the other components.

The fourth capability enables XSEL to provide the user with reasons for its decisions. For the most part, when the user asks why a component was ordered, he has one of four things in mind:

(1) Why was a particular variation of a component selected rather than some other variation?
(2) Why was a particular subtype selected rather than some other subtype?
(3) Why is a particular component type necessary for his applications?
(4) Why does he need the quantity of a component that XSEL has indicated he needs?

The form in which the user asks the question typically indicates which of these four pieces of information he has in mind. The first three questions are all handled by the rules in one context; the fourth question is handled by the rules in a different context.

To enable itself to answer the first three types of questions, whenever XSEL asserts a working memory element specifying that a component of some type is needed or specifying constraints on the allowable subtype or variation, it also asserts a working memory element containing the reason. Since the rules that generate these assertions implicitly contain the reason (i.e., have a set of conditions that define when a particular type or subtype or variation is required), the assertion is simply a re-representation of the reason in a declarative rather than a procedural form. By putting the reasons into a declarative form, other rules – rules that comprise XSEL's knowledge of what constitutes a good explanation – can construct a satisfactory answer to any of the first three types of questions. Constructing such an answer could, for example, involve collapsing a set of reasons into a single more general reason. Or it could involve suppressing certain reasons that might distract the user's attention away from more important considerations. Or it could involve justifying some selection on the basis of its similarlity to an already explained selection.

To answer the fourth type of question, XSEL uses the sets of elements that define its computations. Once XSEL has performed the computation required to determine what the total capacity of some component type should be, its working memory contains elements specifying the value of the intermediate as well as the primitive terms in whatever formula it used. It can use this information to focus the user's attention on significant factors in the computation. If a user asks why

he needs quantity Q of some component X, it is unlikely that he would be satisfied with the following answer:

- You require a total capacity TC of component type x.
- Q of X provide TC.
- Thus you need Q of X.

What the user actually does want to know, however, depends to a considerable extent on the particular situation. If the user wants to know why he needs 2 MS780-DD (i.e., 4 megabytes of primary memory) and if 3 megabytes are needed for one of his applications and 1 megabyte is sufficient for his other six applications, then an important piece of information for the user to have is that one application requires 3 megabytes of memory. XSEL could supply this piece of information by noticing that one of the seven values whose sum specified the total amount of primary memory required is significantly larger than any of the other six values.

The other use a user can make of the fourth capability is to find out why particular configuration constraints were generated. The problem here is essentially the same as the problem of explaining why a component of a particular type, subtype, or variation was ordered, and XSEL uses the same solution. Whenever it recognizes that it has information that implies some configuration constraint, in addition to generating the constraint, it asserts the reason for the constraint. This reason simply describes ths situation that implies the constraint.

The fifth capability listed above is the capability of justifying the reasoning that led to some quantity of components of a particular type being ordered. This capability provides a second level of explanation in those cases in which what is not understood is why the intended uses of the system imply some capacity of some component type. The problem here is to justify the formula used to compute the capacity required. This justification is accomplished by treating the formula as if all of its primitive terms were constants – where the constants are just those values entered by the user. The strategy is to make the unfamiliar formula familiar by reducing it to a formula tailored to the situation at hand.

CONCLUDING REMARKS

The design and implementation history of XSEL is significantly different from that of R1. R1 is the first knowledge-based program to be used at DEC, and thus before it could become part of the culture, it had to prove itself; it had to be an accomplished configurer before DEC would commit itself to exploring its potential. But now that R1 is established at DEC, and now that DEC has a group capable of supporting knowledge-based programs, a different design and implementation strategy is possible. Using R1's capabilities as a base, other programs can be developed and put to work before they are truly accomplished in their domains. Through DEC is not yet ready to design such systems, it does have

the capability necessary to oversee their development from the prototype stage to maturity.

Though the XSEL program is a more ambitious effort than R1 was (in terms both of the variety of the demands of its task domain and of the amount of knowledge it will ultimately have), it will become useful long before it reaches maturity. XSEL is sufficiently developed that within a few months it will be able to be used, in conjunction with R1, to aid salespeople in entering orders and in determining the precise set of components that a customer needs. As knowledge is extracted from configuration design experts, it will be given to XSEL. We expect that by the summer of 1982, XSEL will be sufficiently expert in configuration design and in explaining how configuration design decisions are reached, that it will be of real assistance to the DEC sales force.

ACKNOWLEDGEMENTS

The development of XSEL is being supported by Digital Equipment Corporation. The research that led to the development of OPS5, the language in which XSEL is written, was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, and monitored by the Air Force Avionics Laboratory under Contract F33615-78-C-1151. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Digital Equipment Corporation, the Defense Advanced Research Projects Agency, or the U.S. Government. VAX-11 and PDP-11 are trademarks of Digital Equipment Corporation.

Tom Cooper and Barbara Steele have contributed significantly to both the design and development of XSEL. Other colleagues at CMU have provided valuable suggestions and criticisms, in particular, Jon Bentley, Charles Forgy, and Allen Newell. Sam Fuller, Arnold Kraft, Dennis O'Connor, and many others at DEC have been a constant source of encouragement.

REFERENCES

Clancey, W. J. (1979). Dialogue management for rule-based tutorials. *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, (Tokyo), pp. 155–161.

Forgy, C. L. & McDermott, J. (1977). OPS, A domain-independent production system language. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pp. 933–939. (Cambridge, Mass). Pittsburgh: Dept. of Computer Science, Carnegie-Mellon University.

Forgy, C. L. (1980). *The OPS5 user's manual*. Technical Report, Pittsburgh: Carnegie-Mellon University, Department of Computer Science.

McDermott, J. (1980a). *R1: a rule-based configurer of computer systems.* Technical Report, Pittsburgh: Carnegie-Mellon University, Department of Computer Science.

McDermott, J. (1980b). R1: an expert in the computer systems domain. *Proceedings of the 1st Annual National Conference on Artificial Intelligence*, (Stanford). pp. 269–271.

McDermott, J. and Steele, B. (1981). Extending a knowledge-based system to deal with *ad hoc* constraints. *Proceedings of the 7th Joint International Joint Conference on Artificial Intelligence*, (Vancouver) pp. 776–781.