R1
The Formative Years

by
John McDermott
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15221

R1 is a rule-based program that configures VAX-11 computer systems. Given a customer's purchase order, it determines what, if any, substitutions and additions have to be made to the order to make it consistent and complete and produces a numnber of diagrams showing the spatial and logical relationships among the 90 or so components that typically constitute a system. The program has been used on a regular basis by Digital Equipment Corporation's manufacturing organization since January of 1980. R1 has sufficient knowledge of the configuration domain and of the pecularities of the various configuration constraints that at each step in the configuration process, it simply recognizes what to do; thus it requires little search in order to configure a computer system.

The approach R1 takes to the configuration task and the way its knowledge is represented has been described elsewhere [McDermott 80a, Mc Dermott 80b]. This article provides a detailed description of R1's design and implementation history. As will become apparent, only a part of the effort to develop an expert configurer for Digital dealt in a direct way with AI issues. The other, and in fact larger, part involved providing the ever widening circle of individuals who came in contact with the program with an understanding of why AI tools are appropriate for the task and why the use of AI tools demands a change in attitude toward the programming enterprise.

The development of R1 was supported by Digital Equipment Corporation. The research that led to the development of OPS4 and OPS5, the languages in which R1 is written, was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, and monitored by the Air Force Avionics Laboratory undeı 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

R1's Development

R1's development went through six distinct stages. During the first stage, December 1978 through April 1979, a strategy for attacking Digital's configuration problem was formulated and a program with minimal knowledge of the configuration domain was written to demonstrate the potential of a knowledge-based approach. During the second stage, May 1979 through September 1979, a large amount of knowledge was added to the program. October and November 1979 constituted the validation stage; Digital put R1 through an extensive test and determined that it was sufficiently expert to be used on a regular basis to configure VAX-11/780's. The fourth stage, January 1980 through May 1980, was the period during which the question of how to integrate R1 into Digital's organizational structure was seriously addressed for the first time. During the fifth stage, June 1980 through December 1980, the organizational plans that evolved during the previous stage were implemented. The sixth stage, which began in January 1981 and has not yet ended, is the period of Digital's initial self-sufficiency with respect to R1. In this section, some of the more significant events that occurred within each stage will be described and the main lesson or lessons learned during each stage will be discussed.

Initial Contact

Digital differs significantly from most other computer manufacturers in the degree of flexibility it allows its customers in component selection; rather than marketing a number of "standard systems" with a limited number of options, Digital markets processors with a relatively large number of options, and allows the customer to tailor a system to his needs. One result of this strategy is that most of the systems it sells are one of a kind, and consequently each poses a distinct configuration problem. Since configuring computer systems is time-consuming, Digital made several efforts to develop a computer program that could perform the task. None of these attempts were successful, and in retrospect, it seems clear that the lack of success was due to the fact that the task is knowledge intensive and thus extremely difficult to program with traditional software tools.

Early in 1978 Sam Fuller, who was then a faculty member in the Computer Science Department of CMU, accepted a position at Digital. As he became aware of the problem that Digital was having with automating the configuration task, it occurred to him that a knowledge-based approach to the task might make the problem tractable. In the Fall of 1978, he proposed both to Herb Shanzer, the manager at Digital responsible for solving the configuration problem, and to the production system group at CMU that they meet to discuss the possiblity of CMU developing a program that could do the configuration task. Mike Rychener and I went to Digital in December of 1978 to meet with Shanzer. After spending half a day being told about the nature and scope of the configuration problem, Rychener and I agreed that the problem had characteristics which made a knowledge-based approach appropriate. We proposed that Digital support an effort at CMU to develop a configuration expert Shanzer wanted to believe that we could develop such a program, but didn't (at least not enough to provide funding). We then offered to develop a simple prototype system within a few months that would demonstrate the feasability of a knowledge-based approach. After some discussion, it was decided that we should focus our attention on the problem of configuring VAX-11/780 systems (rather than tackling the bigger and more serious problem of configuring PDP-11 systems). Shanzer offered to make whatever information we needed available to us and appointed Dick Caruso, an engineer who had worked on the configuration problem, to coordinate the information transfer After returning to CMU, Rychener and I decided to take a two-pronged approach to the problem. Rychener would work on developing a program with a general understanding of computer architecture, while I focused on developing the promised prototype. Rychener's work resulted in IPSML, a system that supports the symbolic description and manipulation of computer structures at the PMS (processor-memory-switch) level [Rychener 79].

During December, I spent several days at Digital learning the rudiments of the configuration task. Caruso had done a great deal of work in attempting to uncover the structure of the task. He had broken the task up into a number of smaller subtasks and had generated descriptions of the kinds of actions called for within each subtask. After I had a reasonable understanding of the structure of the configuration task, Caruso gave me two VAX-11/780 configuration manuals which contained a wealth of information about the details of the task After discussing the content of the manuals with him, I went back to CMU to build the prototype configurer.

The issue of what language to use to implement the configuration program was never in question. One of the appeals of the configuration task was that it would give us an opportunity to build an OPS production system for a non-toy task. The OPS language has been described at length elsewhere [Forgy 81, Forgy 77]; it is a general-purpose, rule-based language which 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 satisfied on successive cycles OPS does not impose any organization on rule memory; all rules are evaluated on every cycle If more than one rule is satisfied (or can be instantiated in more than one way) on a given cycle, OPS uses a set of conflict resolution strategies to determine which rule to apply From R1's point of view, it often makes no difference which rule is applied; R1 does, however, rely heavily on the special case strategy. Given two rule instantiations, one of which contains a proper subset of the data elements contained by the other, OPS will select the instantiation containing more data elements on the assumption that it is specialized for the particular situation it is in In OPS4, the version of OPS used to implement the initial version R1, condition elements are represented as lists of constants and and variables.

There are four characteristics of the configuration task that strongly influenced the design of R1:

- Orders are frequently incomplete; the configurer must add components to an order if prerequisite components are missing.
- The data that a configurer needs consists of descriptions of each of the components on an order (together with descriptions of any components that might need to be added).
- The relatively large number of constraints on how components can be associated are conditional on characteristics of the components and on the ways in which already configured components have been associated.
- The configuration task can be decomposed into a set of loosely coupled, temporally ordered subtasks.

The number of components supported on the VAX-11/780 is quite large (about 400 in December 1978). Since R1 would need access to descriptions of both the set of components on an order and any components it might have to add, it was decided that the descriptions of components should be stored in a data base that R1 could access Each description is a set of attribute/value pairs; R1 can access these descriptions by name or by specifying a partial description of a component. Since the constraints on how components can be associated are conditional on the characteristics of components and on the current state of the configuration, it was decided that descriptions of the components ordered and of partial configurations would be held in working memory and that each of the constraints would be formulated as an OPS4 rule. Since the constraints are subtask specific, the rules would be placed into groups on the basis of the subtask to which they were relevant. This would allow each rule to presuppose certain things about the current state of the configuration, thus reducing the number of condition elements required in each rule. It was assumed that R1's basic strategy would be Generate and Test, that is, that some of the rules associated with each subtask would propose extending a partial configuration in a particular way, and that other rules would examine the proposed extension and either accept or reject it.

Implementing the initial version of R1 was straightforward The information that I had gotten from Caruso about the configuration task (augmented with occasional telephone conversations) was sufficient to write a program that could correctly configure simple orders. This prototype program had about 250 rules and took about three man-months to develop. An English translation of one of the rules is shown in Figure 1. In April 1979 the program was demonstrated at Digital to a group of about 60 people and was received with guarded enthusiasm. Though no one was convinced that the program could be extended so that it could configure really complex orders, the program showed enough promise that Digital decided to support further work on it.

It seems almost unbelievable to me now that we almost focused our initial efforts on the PDP-11 world rather than on the VAX-11/780. Though it did not become clear until later just how fortuitous our choice was, it is worth making the point here since if we had focused on a PDP-11, it is likely that R1 would never have found a home at Digital. The VAX-11 nd PDP-11 configuration tasks, in the abstract, are essentially identical But the PDP-11 problem is more complex for two reasons:

- The number of components that are supported on PDP-11 systems is more than an order of magnitude larger than the number currently supported on the VAX-11; thus tackling the PDP-11 task would have required attending to many more details right from the beginning.
- The constraints on the configuration of VAX-11 systems are more specific (less ambiguous) than the contraints for the PDP-11 systems; thus fewer rules are required for the VAX-11 task.

It has turned out the the VAX-11 task was a perfect size. It requires enough knowledge so that a program that can perform the task is interesting But the amount of knowledge required for the task and the specificity of the configuration constraints is such that the problem does not push very hard on state-of-the-art knowledge-engineering techniques.

```
PUT-UB-MODULE-6

    IF: THE MOST CURRENT ACTIVE CONTEXT IS
            PUTTING UNIBUS MODULES IN THE
            BACKPLANES IN SOME BOX
        AND IT HAS BEEN DETERMINED WHICH
            MODULE TO TRY TO PUT IN A
            BACKPLANE
        AND THAT MODULE IS A MULTIPLEXER
            TERMINAL INTERFACE
        AND IT HAS NOT BEEN ASSOCIATED WITH
            PANEL SPACE
        AND THE TYPE AND NUMBER OF BACKPLANE
            SLOTS IT REQUIRES IS KNOWN
        AND THERE ARE AT LEAST THAT MANY
            SLOTS AVAILABLE IN A BACKPLANE
            OF THE APPROPRIATE TYPE
        AND THE CURRENT UNIBUS LOAD ON THAT
            BACKPLANE IS KNOWN
        AND THE POSITION OF THE BACKPLANE IN
            THE BOX IS KNOWN

    THEN:   ENTER THE CONTEXT OF VERIFYING
            PANEL SPACE FOR A MULTIPLEXER
```

Figure 1. A Sample Rule

From Novice to Expert

Though several people at R1's initial demonstration believed that it was in Digital's interest to support further work on the program. Dennis O'Connor (representing System Manufacturing) emerged as R1's principal sponsor. O'Connor established a steering committee consisting of himself, Sam Fuller (representing Central Engineering), and Lou Reagan (representing Order Processing Administration). Kent McNaughton, a member of O'Connor's group, was asked to coordinate the project.

The magnitude of the configuration problem at Digital created a certain amount of pressure to move beyond the demonstration stage as quickly as possible. Since I lacked an understanding of the complexity of the task, it was difficult for me to estimate just how long it might take to develop the program to a point where its performance was comparable to that of skilled humans. The initial version of R1 appeared to have most of the basic knowledge needed; that is, it could configure simple systems correctly. On the basis of the number and kinds of mistakes it made on more complex orders, I guessed that it had about half the knowledge it needed, and suggested that an adequate version of R1 could be developed within four or five months. After some discussion, that suggestion somehow became a commitment to deliver to Digital, by the end of September, a program that (1) could correctly configure at least 75% of the orders it was given, and (2) could be easily modified or extended to configure any orders that it configured incorrectly. We met this deadline with a day and a half to spare.

The task for those five months actually had two distinct parts. It was clear that a considerable amount of knowledge of configuration constraints would have to be added to the program and that some of the knowledge the program had would have to be modified; this is the part of the task that CMU committed to. The other part of the task was to extend the data base of component descriptions. In May 1979, there were 420 components supported on the VAX-11/780; the data base used by the initial version of R1 had fewer than 100 component descriptions. Caruso was assigned the task of producing a complete data base Since Digital had no single source containing all of the information required (8-10 pieces of information about each component), and since much of the information was not in machine readable form, collecting the information was time-consuming. About three man-months of effort were required to create the complete data base.

The task of extending R1's configuration constraint knowledge proceeded in parallel with the data base creatoin task There were two senses in which R1's constraint knowledge was incomplete. (1) the sets of rules that enabled R1 to perform particular subtasks were sufficient for simple systems, but needed to be augmented to handle systems consisting of components with more complex interrelationships. (2) R1 had no knowledge of how to configure some types of components, and thus needed to acquire sets of rules that would enable it to perform the subtasks implied by those component types. My approach to extending R1's knoweldge base was esentially the same as the approach taken by other expert system builders [Feigenbaum 77]. R1 was given a number of orders to configure. Its output was shown to configuration experts who were asked to evaluate the adequacy of the configurations. When an expert found a problem, we would talk about it. Such discussions always resulted in the characterization of the situations in which R1's actions were inappropriate, and an indication of what actions would be appropriate. Ordinarily, especially in the beginning, a considerable amount of configuration knowledge came to light during each interaction with an expert. Thus after each interaction several rules could be written; one to correct the problem manifestation and a number of others which, though not directly related to the problem at hand, were associated, in the mind of the expert, with the problem situation By the end of September 1979, R1 had a database of 420 component descriptions and its rule memory contained about 750 rules The fact that R1's knowledge tripled during this stage came as something of a surprise. Though it had been clear in May that R1's knowledge was incomplete, still R1 was able to correctly configure systems. It is interesting that R1 needed twice as much knowledge to deal with special situations as it did to perform the basic task.

It became apparent during this stage that the method R1 uses is Match [Newell 69]. As mentioned, an initial assumption had been that R1 would use Generate and Test as its principal problem solving method. Each set of rules included a rule whose function was to test to determine if the current subtask had been accomplished. In the demonstration version of R1, each of these rules was a (vacuous) general case of the other rules associated with the subtask and thus fired only after all other applicable rules had been applied. I assumed that as the system was developed, (1) at least some of these rules would have to be modified so that they could recognize when the goals of their subtasks had been achieved and (2) other test rules would have to be added that could recognize when an incorret configuration was being generated and inititate actions that would get R1 back on the track. As it turned out, neither of these was necessary.

The important lesson learned during this stage is that Match is an appropriate problem solving method for some class of domains. The configuration task is constructive; it has the following characteristics:

- It is possible to impose a partial ordering on the set of components to be configured such that if the components are configured in that order, they can be configured without any backtracking.
- The partial ordering can be defined, dynamically, on the basis of the total set of components to be configured and the associations already made among those components.

Because it is possible to configure a set of components correctly, without backtracking, if enough knowledge is brought to bear at each step to determine which component should be configured next, there is no need within any subtask to test whether a just configured component has been configured correctly. Furthermore, since each subtask extends the configuration in whatever ways are appropriate within the confines of the subtask, there is no need for a rule to determine whether the goal of the subtask has been accomplished. When none of the rules associated with the subtask are satisfied, all that can be done has been done. One positive consequence of the use of the Match method (in addition to the elimination of search) was that adding knowledge to R1, once it was extracted from the experts, was straightforward. Since each rule defines the set of situations to which it is relevant, if a rule is applicable, it should and will fire (unless it is a general case of some more specific rule that fires first and changes the current situation so that the more general rule is no longer applicable).

Validation

During October and November 1979, Digital put R1 through a formal validation procedure. The purpose of this test was to determine whether R1 was expert enough in the configuration task to be used in place of human experts The validation process consisted of giving R1 the 50 most recently booked orders and having a group of six experts carefully examine R1's output for correctness. The experts were given orders in groups of 10; they spent 8 hours on the first order, 2 hours on the second, and then about 1 hour on each of the other 48. After checking a group of 10 orders, they forwarded errors due to incorrect configuration knowledge to me and errors due to inaccurate descriptions in the component data base to Vaidis Mongirdas, the engineer who replaced Caruso when Caruso moved to another job at Digital. The problems were fixed and the incorrect orders rerun with the next group of 10.

The experts were extremely impressed with R1's performance. R1 made 12 mistakes (all of which were easily fixed) in configuring the 50 orders All but two of the mistakes were at a level of detail below that at which humans responsible for configuring systems work out the configurations The team of experts pointed out that in addition to providing more detailed configurations than the human configurers, R1 (even with its still imperfect knowledge) was likely to do a better job than the humans since they are subject to lapses of attention. The conclusion drawn from the validation process was that R1 should begin to be used on a regular basis in at least one of the Final Assembly and Test (FA&T) plants and that its use should be extended to other FA&T plants as soon as this was feasible.

With something less than a man-year's effort, R1 had developed to a point where it could begin to be used in place of human experts. Though R1's task domain was very narrow, the configuration task has a considerable amount of complexity and thus requires a significant amount of knowledge. It is unlikely, I think, that a program using more traditional software tools could be developed in anywhere near the same time frame. Aside from the fact that the task is highly conditional (there are, on the average, three possible paths that could be followed on each of the typically 1000 steps in the task), a program that can perform the task must necessarily be developed incrementally. Given the amount of knowledge required and the fact that the knowledge can apparently be extracted from the experts only as they see from mistakes the program makes what knowledge is lacking, it would seem that any attempt to develop a program that is not strongly recognition driven would be doomed to failure

Organizing to use R1

The concern at Digital during the first year of R1's development was almost exclusively with whether and how well R1 would be able to perform the configuration task. Little attention was paid to the question of how R1 would be integrated into the system, configuration and assembly process. At the end of the validation stage, this question became prominent. There were two issues that had to be attended to:

- It was important for Digital to find a way to begin to use R1 that would disrupt the existing process as little as possible and that would provide a framework within which R1's configuration expertise could grow.
- It was important for Digital to establish a group able to continue the development of R1 and to extend its capabilities to other computer systems (eg, the PDP-11).

The issue of use was relatively easy to deal with. In each of Digital's FA&T plants, people called technical editors configure each system to be built and give their configuration diagrams to technicians who actually assemble the systems in one of Digital's FA&T plants The person who had been the technical editor for VAX-11/780 systems became R1's "supervisor." The role of the supervisor was to examine R1's configuration diagrams for correctness. Correct output was given to the technicians for use in assembly Data base problems were reported to and fixed by Mongirdas; rule problems were reported to and fixed by me during my monthly visits to Digital

The issue of establishing a group to continue the development of R1 was more problematic. Several months were spent in attempting to define the functions that the group would perform, determining where the group should be located within Digital's organizational structure, and searching for a set of individuals, particularly a manager, who could form the nucleus of the group. It became clear that the group would have three principal functions:

- data collection
- program maintenance and development
- process development

The data collection task would involve extending R1's data base of component descriptions—the task that the engineers assigned to help with R1's development had been doing all along. the program maintenance and development task would involve making modifications to R1's rules as inadequacies in its knowledge were discovered and extending R1 so that it could configure systems other than the VAX-11/780. The people performing this task would essentially take over the role that CMU played; to do this, it would be necessary for them to have a solid understanding of the OPS language. The third task, process development, was the least well defined. It was clear that in order for R1 to be an effective tool, it was necessary that it be easily accessible to people in a number of different Digital organizations (eg, manufacturing, engineering, sales). The process development task was to invent ways of facilitating information flow.

While Digital was concerning itself with organizational matters, R1 was reimplemented at CMU. The impetus for the reimplementation was the availability of OPS5, a new version of OPS. OPS5 differs from OPS4 primarily in the way the conditions of a rule can be expressed In OPS4, a condition element is represented as a list of symbols; in OPS5, it may be represented either as a list of symbols or as an object with associated attribute/value pairs. If an attribute name appears in a condition element, it is interpreted by OPS5 as the name of a working memory element field. A condition element is instantiated by a working memory element if each value (constant or variable) in the condition element is equal to or can be bound to the symbol in the working memory element in the specified field OPS5 is a significant improvement over OPS4 both in ease of use and in the intelligibility of the rules.

Since the only difference between OPS4 and OPS5 is in the way in which conditions are expressed, it would have been possible, in a few weeks time, to simply translate the OPS4 rules into OPS5. However, during the course of developing R1, a great deal of knowledge was added that R1 was unprepared for. Although R1 had been able to handle this massive intrusion of new knowledge, it resulted in a considerable amount of redundancy, some cases of many rules where a single general rule would do, and some cases of doing easy things the hard way. Moreover, in the initial version of R1 the knowledge required to perform two subtasks was not represented in the form of rules; instead, LISP routines that could do the subtasks were invoked by rules which recognized situations in which the subtasks were germane. At the time, since the experts swore (falsely, it turned out) that all of the knowledge relevant to these subtasks was on the table, it seemed appropriate to put the knowledge in an algorithmic form. Given these problems, it appeared that a serious reimplementation effort would be likely to have the effect of making clearer precisely what knowledge R1 had, and thus make the task of maintaining and extending R1 easier.

Brigham Bell, Barbara Chessler, and Tom Cooper worked with me at CMU on the reimplementation of R1; the effort took about four man-months. The resulting program, though it had some capabilities that the OPS4 version lacked, consisted of about 500 rules (two-thirds the number that the earlier version had). About 100 rules, those that tested to determine whether a goal had been achieved, were simply eliminated. Another 50 rules, those that accessed the data base to gain further information about a component, could be eliminated because OPS5's attribute/value representation makes it possible to match against only part of a component description; thus each relevant description could be accessed once and stored as a single working memory element. Since about 25 rules were added to the new version to provide additional capabilities, and another 25 to replace the functionality of the LISP routines, about 150 rules were eliminated by reducing redundancy, finding appropriate generalizations, and simplifying configuration strategies.

I was surprised both by the length of time the reimplementation took and by the significantly smaller size of the new version It appears necessary in domains such as the configuration domain to develop programs incrementally. But incremental addition of knowledge (given our current ignorance of how to build programs that can learn) is likely to result in a knowledge base that is convoluted and thus, from a human maintainer's point of view, less intelligible than it might otherwise be. Once much of the knowledge relevant to a domain has been extracted from experts, significant gains in intelligibility can be achieved by reimplementing programs on the basis of the more complete picture. It is not clear in R1's case how many reimplementations will be necessary. But as will become evident below, it is likely that the number is greater than 1.

A Foundation to Build On

By May 1980, the functions and structure of the group that was to be responsible for the development of R1 had been defined. Arnold Kraft was selected to manage the group which was to consist, during its first year, of 12 people. In addition to Kraft, there were to be three people responsible for data collection, five responsible for program maintenance and development, and three responsible for process development. By August, seven of these people had been hired and the rest were in place by December. The people responsible for data collection and for process development needed no special training. But the people responsible for program maintenance and development had to become proficient in OPS5. These people had a variety of backgrounds and amounts of programming experience (from almost none to much); one of them, John Ulrich, had a strong background in AI. On the average, it took each one of them about three months to become proficient in OPS5. Given three months of practice with the language, they were able to make appropriate modifications and extensions to the rules whenever inadequacies in R1's knowledge manifested themselves. During this period, R1 began to be used in all of Digital's FA&T plants in the United States. In October, two members of the group, Ed Orcuich and Bill Brodie, extended R1's capabilities so that it could configure VAX-11/750's as well as VAX-11/780's. Though much of the knowledge needed to configure the two systems is the same, there are a number of significant differences in the onfiguration constraints. Thus this extension to R1 demonstrated a solid grasp of knowledge engineering techniques.

One of the first actions of the people responsible for process development was to formalize the reporting of problems with R1. A problem report form was developed. Whenever the people in the FA&T plants responsible for supervising R1 believed that a system had been incorrectly confiugred, they would fill out this form and send it to the group responsible for maintaining R1; during the period from May 1980 through the end of the year, report forms were generated for about 40% of the orders processed. When a report form was received, it was given to Mongirdas; his task was to determine whether the configuration produced by R1 was in fact incorrect and if so to determine the cause of the problem. The problems encountered distributed themselves fairly evenly among the following five classes:

- An incorrect component description in the data base.
- Incorrect configuration knowledge.
- Incomplete configuration knowledge.
- An error in the data input to R1.
- A confusion on the part of the person reporting the problem (ie, a non-problem).

When the problem had been categorized, it was given to one of the members in the group for action. The data base problems were fixed by the people responsible for data collection. The problems of incorrect rules and missing knowledge were addressed by the people responsible for maintaining the program (working in conjunction with the engineers responsible for data collection). The problems of dealing with report forms that should not have been generated and with errors in input were the responsibility of people in charge of process development. Many people were disturbed by the poor performance of R1 over this eight month period; the expectation after the validation stage was that R1 would soon be configuring at least 90% of the orders correctly. But in retrospect, it is clear that at the end of the validation stage R1 was still a very inexperienced configurer. It had encountered only a tiny fraction of the set of possible orders, and consequently its knowledge was still very incomplete.

In July 1980, a manager at the FA&T plant using R1 mentioned that R1 was a less helpful tool than it might be because whenever it encountered an order that was incomplete, it added whatever components were necessary to make the system complete. This complaint came as something of a shock since one of the principal reasons for developing R1 was to provide this capability. The problem, it turned out, was that when an FA&T plant receives an incomplete order, it assembles whatever subset of components it can without adding anything to the order and at the same time contacts the customer to find out whether the remaining components should be shipped as "spares" or the missing components added. Since R1 always added missing components, its output made evident which orders were incomplete (about 20% of the orders received), but could not be used in their assembly. To make R1 more helpful, it was modified so that whenever it configured an incomplete order, it kept track of what components on the order required other components to be added and then configured the order a second time after marking those components with missing prerequisites as not to be configured. This extension, which I worked on with John Barnwell and Ed Orciuch at Digital, took less than a man-month.

After being provided with this two-pass version of R1, the FA&T plant asked that R1 be modified so that it could handle "project orders." Most of the systems configured in the FA&T plants are configured in the absence of any information about how the customer intends to use his system. Thus R1's configuration rules assume "typical use." Some customers, however, sometimes order a large number of almost identical systems, and in such cases it is feasible to tailor the configuration of these systems to fit the particular needs of the customer. There is a group at Digital responsible for generating a set of guidelines for configuring such systems, but R1 was incapable of modifying the configurations it produced on the basis of such guidelines. During October and November 1980, work was done at CMU to provide R1 with that capability [McDermott 81a]. The strategy we adopted was to leave untouched, for the most part, the rules comprising R1's basic configuration capability. This left R1 performing the configuration task, in the absence of customer-specific constraints, in the same way it always had. We simply added a set of rules that recognize situations in which there is a customer-specific constraint and components or partial configurations that indicate it is time to attend to that customer-specific constraint. Each of these rules modifies working memory elements in a way that insures the satisfaction of the constraint. In a sense, these new rules are disassociated from the main-stream configuration task; they stand outside and allow customer-speicific constraints to step in, change the world in some appropriate way, and then step back out and let R1 continue about its business.

One important capability that the initial version of R1 lacked was that of designing the floor layout of cabinets and other free-standing components R1 simply assumed an indefinitely long room, and laid the components out in a straight line. This made it impossible for R1 to determine precisely the lengths of some of the cables needed to connect pairs of components. The reason for delaying the implementation of a more adequate floor layout capability is that nothing of interest can be done without information about the room or rooms that will house the system. But as plans began to be developed to move R1 out of the FA&T plants and into the sales offices (where the user of the program has access to information about the customer's site), implementing a more adequate floor layout capability became a reasonable goal. About three months' work by Brigham Bell and Barbara Steel at CME during the Fall of 1980 resulted in such a capability. R1 now allows a user to provide room information and specify whatever constraints on component placement he wishes. Then using a few general heuristics about busy doors and component orientation and its knowledge of cabling constraints, R1 searches for a way of laying out the components that satisfies whatever constraints the customer has imposed.

The activity during this stage of R1's development resulted in a significant increase in the size of its knowledge base; in June 1980, R1's knowledg consisted of about 500 rules. The implementation of the two-pass capability added another 50 or so rules, the customer-specific constraint capability added about 100 rules, and the floor layout capability another 100 The rules that were needed to supply the missing knowledge brought to light by the problem reports added approximately another 100 rules. Thus by December 1980, R1's knowledge base had grown to about 850 rules.

To this point, I have said nothing about the amount of time R1 takes to configure an order. Since processing time has continually been of some concern, it is worth attending to briefly. at the end of the validation stage, the average timer required to configure an order was about 2.5 cpu minutes on a PDP-10 (version KL). By June 1980, the complexity of the average order (number of components) had increased significantly and the average time to process an order grew to about 4 minutes. When the two-pass capability was added, since about a fifth of the orders had to be run a second time, the average time grew to about 5 minutes In August, R1 moved from a PDP-10 to a VAX-11/780; this resulted in an additional factor of 3 increase—to about 15 cpu minutes per order. For a variety of reasons, the view at Digital since early in R1's development was that it was important that the average time to configure an order be reduced to less than 2 minutes. To achieve that goal, Charles Forgy, the principal designer of OPS, had begun working early in 1980 on a BLISS version of the OPS5 interpreter. OPS has two distinct parts, it consists of a rule compiler and an interpreter. Until this point, all versions of OPS had been implemented in LISP (MACLISP for the PDP-10 and FRANZLISP for the VAX-11). Though a BLISS version of the interpreter was finished by October 1980, Digital decided not to use this interpreter until it had implemented a BLISS version of the rule compiler Though the BLISS version of the compiler will not be finished for a few more months, it appears from the timing studies that have been done that it will take about 1.5 minutes of cpu time of a VAX-11/780 to configure an average order.

The fact that R1's knowledge grew at least as much during this stage of its development as in any of the previous four stages, points up how important it is for a knowledge-based program to be open to new and often altogether unexpected demands. One of the claims frequently made for rule-based languages is that they support the incremental growth of programs. This seems to be born out in R1's case. Both the two-pass capability and the customer-specific constraint capability placed demands on R1 that had in no way been planned for; yet both capabilities were easily accommmodated with virtually no change to the rules already there. The extensive growth of R1's configuration knowledge (in the second stage as well as the fifth) was likewise accomplished without major modifications to existing knowledge. The error reporting process proved to be an extremely valuable means of spotting knowledge inadequacies. It essentially provides a way of continuously testing R1, and such continuous testing is clearly particularly important during the early life of an expert system when its knowledge base is nowhere near adequate and while the task that it is supposed to do is being redefined and enlarged.

Self-Sufficiency

During the two years from December 1978 to December 1980, almost all of the development work on R1 was done at CMU; the principal exceptions are the creation of the component data base and the implementation of the version of R1 that configures VAX-11/750's. Until January 1981, Digital did not really have the proficiency required to do more than maintain the existing system. At that point, however, the group was ready; it is planned that all further work on R1 will be done at Digital by Digital personnel. Given R1's newfound reliability, the focus at Digital has turned to the problem of extending R1 so that it can configure PDP-11 systems. Because the various PDP-11's are less constrained in certain respects than the VAX-11's and because the number of components supported on PDP-11's is significantly greater than the number currently supported on the VAX-11's, the task of extending R1 so that it can configure PDP-11's will be challenging. (Another piece of work, started a few months ago by John Ulrich and Kalman Reti at Digital, is a research effort aimed at evaluating an alternative representation of R1's knowledge. Their program, which should be finished in a few months, is essentially a frame-based representation of configuration knowledge. It will be interesting to compare the relative strengths and weaknesses of the rule-based and frame-based approaches.)

Though CMU is no longer directly involved with the development of R1, we will be working on two tasks that lie adjacent to the configuration domain. One of these tasks is assisting salespeople with component selection. XSEL, the program being developed, will interact with a salesperson to obtain the information required in order to tailor a system to fit the customer's needs. After obtaining this information, it will select a CPU, some amount of primary memory, some software, and whatever devices (eg, disk drives, tape drives, terminals, printers) the customer needs; this skeletal order will then be passed to R1 to be fleshed out (with cabinets, boxes, backplanes, controllers, cables, etc) and configured. During the interaction with the customer, it may become evident to XSEL that the particular needs of the customer imply special configuration constraints; if so, XSEL will inform R1 of those constraints [McDermott 81b]. The other task is that of managing the scheduling of orders. When ISA, the program which will perform this task, is ready for use, XSEL will be able to take into account the effect that its choice of components has on the date on which the system can be delivered.

Though R1 is still in its adolescence, its palce in Digital not only looks secure, but it appears that complementary knowledge-based systems may spring up on all sides. The group responsible for R1 has demonstrated that it can provide a useful service. Early in R1's development, few at Digital believed that R1 would ever amount to anything. As it developed, a scattering of individuals became convinced of its potential. But it was not until recently that a significant number of people at Digital have given much credence to knowledge-based systems. Within the past six months, however, the number of people at Digital with confidence in knowledge-based techniques has increased dramatically.

How R1 Didn't Sink

I understand much more clearly now than I did two and a half years ago what must happen in order for an AI application to be "successful." When I was introduced to Digital's configuration problem, I was sure that it would not be to difficult to develop a knowledge-based program to solve be too difficult to develop a knowledge-based program to solve the problem. When I shared this insight with people at Digital and encountered skepticism, I was not terribly surprised. Having never encountered the rhetoric of the knowledge engineer, they could scarcely be expected to appreciate knowledge's potential. But after developing the demonstration version of R1, I was surprised when the skepticism was replaced, not with belief, but with caution At each stage in R1's development, I looked for the caution to disappear. And at each stage a few individuals became less cautious. But only recently have a significant number of people begun to believe that knowledge-based programs have a future at Digital. In retrospect, what is surprising is that R1 managed to stay afloat in such a sea of caution. That it did is, I think, due to two, partly fortuitous circumstances: (1) At each stage in its development, R1 convinced a few people who were in a position to assist in its development that it had real promise. (2) Only occasionally, and very locally, did R1 do less than was expected of it; thus it never made any enemies.

The first person to assist in R1's development was Sam Fuller. He brought the production system group at CMU into contact with the configuration problem. But more importantly, he brought us into contact with a problem that was causing a large number of people a large amount of grief. I think it is clear that part of the reason R1 convinced some people that it had promise is that they wanted so badly to believe the configuration problem could be solved. Yet despite the seriousness of the problem, no one at Digital was prepared to fund the development of a knowledge-based configurer until after the demonstration version of R1 had been implemented. Thus if the CMU Computer Science research environment were not structured in a way that permits speculative efforts, such as the initial version of R1, to be supported, work on R1 would never have begun. Once the demonstration took place, a number of people at Digital committed themselves to making R1 a success. Besides being directly concerned with the configuration problem, these people, in general, believed that it was important to explore the potential of new software technologies; thus these people were predisposed to believe that the new and unfamiliar might be good. Finally, as R1 became increasinly expert at the configuration task, people more firmly tied to the present (ie, people who believed that a bird in the hand, no matter what, is worth two in the bush) began to believe that R1 had real promise.

If R1's various supporters had not emerged when they did, R1 could have easily just sunk out of sight. But for R1 to survive, not having enemies was as important as having some strong supporters. R1's place in Digital was tenuous enough that if a few people had believed that exploring R1's potential was a serious mistake, the exploration would have stopped. Three factors kept the all-pervasive caution for turning to hostility. (1) The task of developing a program that could configure VAX-11/780's was, as mentioned, of just the right degree of difficulty. R1 was able to devleop at a reasonable rate and so anyone who looked could see progress. (2) The number of people immediately involved with R1 was quite small at first and grew very gradually. Those closest to R1 were for the most part those who believed it had promise, and thus were willing to shut their eyes when R1 stumbled. (3) Finally, the people who were spokesmen for the project, Kraft, McNaughton, and O'Connor, worked hard to manage people's expectations to insure that no one would count on more from R1 than it could deliver.

Though R1 now has a large number of strong supporters at Digital and the extreme caution toward knowledge-based programs is waning, I have one remaining, quite general concern. It is not clear that all (or even most) of R1's supporters realize that R1 will always make mistakes. The problem is that at least some of R1's supporters think of it as a program rather than as an expert. There is, of course, a big difference between programs and experts. Finished programs, by definition, have no bugs. When experts are finished, on the other hand, they're dead. During the last two years, I have hammered on the theme that a knowledge-based program must pass through a relatively lengthy apprenticeship stage and that even after it has become an expert, it will, like all experts, occasionally make mistakes. The first part of this message got through, but I suspect that the second has not. My concern, then, is whether, as this characteristic of expert programs is recognized, Digital (or any large corporation) will be emotionally prepared to give a significant amount of responsibility to programs that are known to be fallible. ■

Acknowledgements

Many people, in addition to those mentioned above, helped with R1's development The people who have provided significant assistance include Jim Baratz, Jon Bentley, Andrea Crowe, John Dennis, Lou Gaviglia, Caroline Hayes, Betsy Herk, John Holman, Dave Kiernan, Allan Kent, Linda Marshall, Paul McManus, Allen Newell, Mike Powell, Mike Renda, Mary Sack, Diane Secatore, Ed Siegmann, Mark Stevenson, and Peter Zotto

References

Feigenbaum 77. Feigenbaum, E A., The art of artificial intelligence In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pp 1014-1029 MIT, 1977

Forgy 77. Forgy, C. L. and J McDermott, OPS, A domain-independent production system language. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pp 933-939. MIT, 1977.

Forgy 81. Forgy, C. L., The OPS5 user's manual. Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

McDermott 80a McDermott, J., R1: a based configurer of computer systems Technical Report, Carnegie-Mellon University, Department of Computer Science, 1980

McDermott 80b McDermott, J , R1 an expert in the computer systems domain In *Proceedings of the 1st Annual National Conference on Artificial Intelligence*, pp 269-271 Stanford University.

McDermott 81a. McDermott, J and B Steele. Extending a knowledge-based system to deal with ad hoc constraints. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence* University of British Columbia, Vancouver, British Columbia, Canada, 1981.

McDermott 81b McDermott, J., XSEL a computer salesperson's assistant In J. Hayes and D Michie (editors), *Machine Intelligence 10*, forthcoming, 1981.

Newell 69. Newell, A, Heuristic programming: ill-structured problems. In J S Aronofsky (editor), *Progress in Operations Research*, pp 361-414 John Wiley and Sons, 1969

Rychener 79 Rychener, W., A semantic network of production rules in a system for describing computer structures. Technical Report, Carnegie-Mellon University, Department of Computer Science, 1979