AI Magazine Volume 5 Number 3 (1984) (© AAAI)

R1 Revisited: Four Years in the Trenches

Judith Bachant
Intelligent Systems Technology Group
Digital Equipment Corporation
Hudson, Massachusetts 01749

John McDermott
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

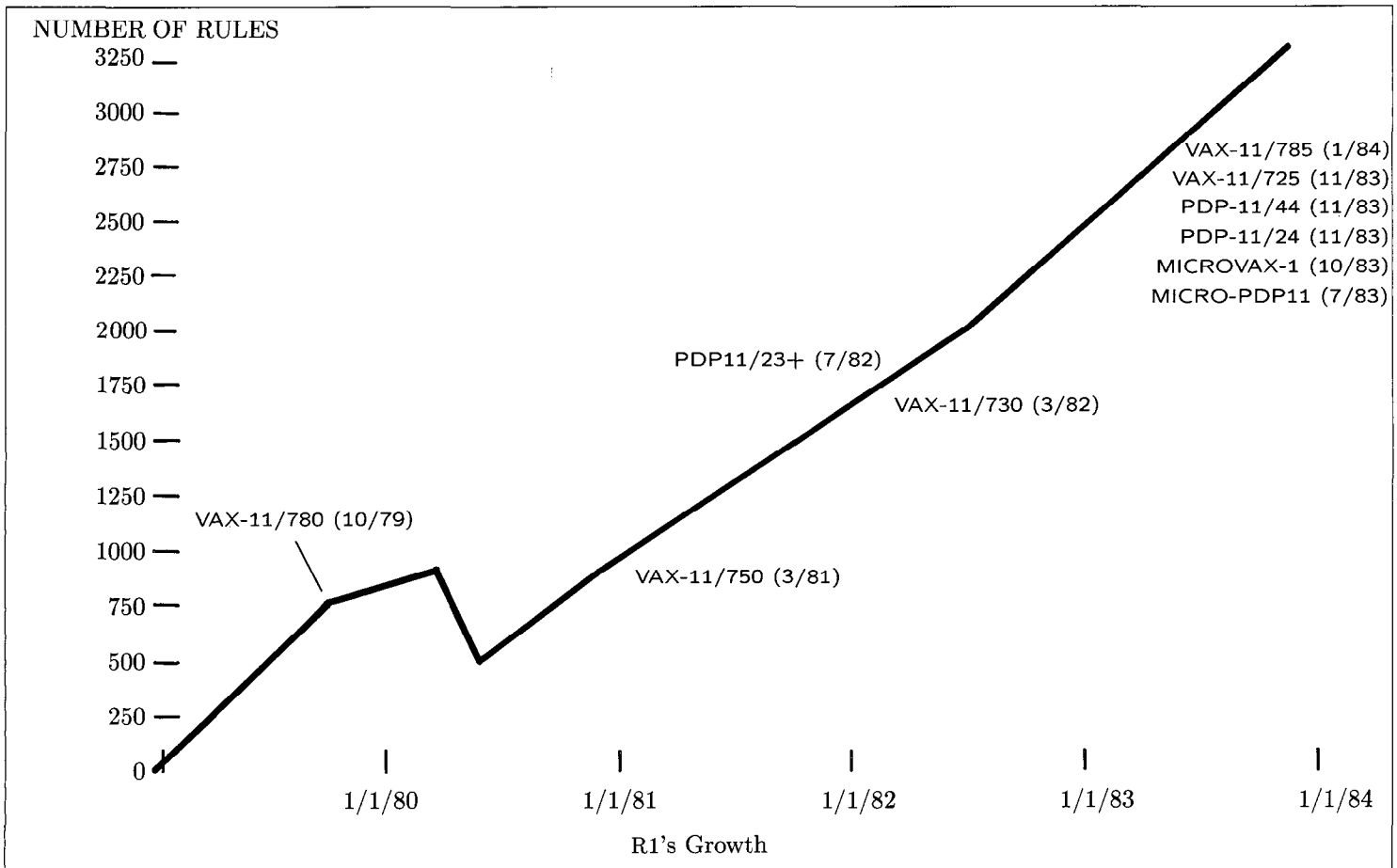
In 1980, Digital Equipment Corporation began to use a rule-based system called R1 by some and XCON by others to configure VAX-11 computer systems In the intervening years, R1's knowledge has increased substantially and its usefulness to Digital continues to grow. This article describes what is involved in extending R1's knowledge base and evaluates R1's performance during the four year period.

IN THE SUMMER 1981 ISSUE of the AI Magazine, an article entitled "R1: the formative years" described how a rule-based configurer of computer systems had been developed and put to work (McDermott, 1981). At the time that article was written, R1 had been used for only a little over a year and no one had much perspective on its use or usefulness. R1 has now been configuring computer systems for over four years. This experience has provided some insight into the ease and difficulty of continuing to grow an expert system in a production environment and into the kind of performance expectations it might be reasonable to have about a current generation rule-based system.

The approach R1 takes to the configuration task and the way its knowledge is represented have been described elsewhere (McDermott,1980) and (McDermott, 1982). Briefly, given a customer's purchase order, R1 determines what, if any, substitutions and additions have to be made to the order to make it consistent, complete, and produce a number of diagrams showing the spatial and logical relationships among the 50 to 150 components that typically constitute a system. The program has been used on a regular basis by Digital Equipment Corporation's manufacturing organization since January, 1980. R1 has sufficient knowledge of the configuration domain and of the peculiarities of the various configuration constraints that at each step in a configuration task it is usually able to recognize just what to do; thus it ordinarily does not need to backtrack when configuring a computer system.

At the beginning of R1's development, no clear expectations existed about how long it would take to collect enough knowledge to make R1 an expert. We did expect that at some point the rate at which R1 would acquire new knowledge would at least slow, if not stop. We even thought that R1 would be done eventually (that is, R1 would enter a maintenance mode of well-defined and minor additions, interspersed with occasional bug fixes.) It is difficult now to believe R1 will ever be done; we expect it to continue to grow and evolve for as long as there is a configuration task. It may be that if R1's domain were less volatile, R1 would not require perpetual development. But it is

A large number of people have played critical roles in R1's development. Among those who deserve special mention are John Barnwell, Dick Caruso, Ken Gilbert, Keith Jensen, Allan Kent, Dave Kiernan, Arnold Kraft, Dennis O'Connor, and Ed Orciuch. We want to thank Allen Newell, Dennis O'Connor, and Ed Orciuch for their helpful comments on earlier drafts of this article



R1's Growth

Figure 1.

probably also true that if the domain were less volatile, the task would not require a knowledge-based system.

The early expectations about R1's performance were likewise vague, except just as R1 was beginning to be used, a Digital employee responsible for the configuration process predicted that for R1 to be useful, 90% to 95% of its configurations would have to be perfectly correct. This performance goal is interesting, not so much because R1 took three years to reach it, but because it turned out to be completely wrong. R1's task is just one small part of a process designed to ensure that high quality computer systems are built. Significant redundancy exists in the process precisely because historically no individual has both known enough about configuration and been able to pay close enough attention to each order to be entrusted with the total responsibility. R1 was able to provide significant assistance even when it knew relatively little because the people who used R1 did not demand more of it than of its human predecessors. The one definite performance expectation almost everyone had about R1 in its early days was that it would always configure the same set of components in the same way. It is obvious now and should have been obvious then that this expectation could have been satisfied only if R1 had been discouraged from becoming more expert.

These expectations about R1's developmental and performance histories introduce the two parts of the article. In the next section, the focus will be on the kind of involvement required to extend R1's knowledge base. The final section's focus will be on the kinds of erroneous behavior R1 has exhibited.

R1's Developmental History

This section provides a somewhat anecdotal trip through R1's past. Although it mentions the first year, when most of the activity was at Carnegie-Mellon University [CMU], the primary focus is on the four following years, after R1 began to be used at Digital. When CMU handed over the initial version of R1 to Digital in January 1980, Digital scrambled to put an organization in place that could continue its development. This organization, currently known as the Intelligent Systems Technologies group, began with only five individuals, none of whom had any background in AI. Over the past four years, the group has grown to 77 people responsible for eight different knowledge-based systems, one of which is R1. As R1 was developed, an attempt was made to effect a division of labor between those people responsible for representing R1's knowledge and those responsible for collecting and validating that knowledge. Of the initial technical people, one was an engineer who played the roles of both

a domain expert and of an interface to other domain experts outside the group; the other three people took the knowledge collected by the engineer and formulated it so it was compatible with R1's other knowledge. When the organization was a little over two years old the technical group had grown to eight people, five of whom were responsible for encoding the knowledge collected and validated by the other three. The size of the R1 technical group is still about eight. Now, however, less of a distinction exists between the people responsible for knowledge encoding and those responsible for knowledge collection.

The Knowledge R1 Acquired

Over the past four years, the amount of effort devoted to adding knowledge to R1 has remained relatively constant at about four worker-years per year. And R1's knowledge has grown at a relatively constant rate, though the focus has shifted around. At times the task of eliminating inadequacies in R1's configuration knowledge has received the most attention; at other times, the group's energies have been directed primarily at broadening R1's abilities in various ways. Figure 1 shows the rate at which R1's knowledge has grown; the points in time at which R1 became able to configure new system types are marked. Figure 1 does not show the amount of product information to which R1 has access. This information, which is stored in a data base, is a critical part of the body of information needed to configure a computer system correctly. R1 retrieves the description of each component ordered before it begins configuring a system; while configuring the system, if it determines some piece of required functionality is missing, it searches the data base for components that will provide that functionality. R1 currently has access to almost 5500 component descriptions. We do not have good data on the rate at which the data base has grown, but what data we have suggest the growth rate is quite irregular.

In this article, R1's growth is measured in number of rules. The following values hint at the amount of knowledge an R1 rule contains. The average conditional part of one of R1's rules has 6.1 elements (the minimum number is 1 and the maximum 17). Each element is a pattern that can be instantiated by an object defined by as many as 150 attributes. On the average, a pattern will mention 4.7 of those attributes (the minimum is 1 and the maximum 11) and restrict the values which will satisfy the pattern in various ways. The tests are mostly simple binary functions that determine whether some value in the object has the specified relationship to some constant or to some other value in that or another object. The action part of an average rule has 2.9 elements (the minimum is 1 and the maximum 10). Each element either creates a new object or modifies or deletes an existing object. A rule can be applied when all of its condition elements are instantiated.[1]

[1]For additional information about the nature of R1's rules as well as those of other systems written in OPS5, see (Gupta, 1983)

Work on R1 began in December 1978. During the first four months, most of the effort was on developing an initial set of central capabilities. The initial version of R1 was implemented in OPS4, a general-purpose rule-based language (Forgy, 1979). By April, R1 had 250 rules. During the same period, a small amount of effort was devoted to generating descriptions of the most common components supported on the VAX-11/780. After this demonstration version of R1 had been developed, most of the effort during the next six months was divided between refining those initial capabilities and adding component descriptions to the data base; in October 1979, R1 had 750 rules and a data base consisting of 450 component descriptions. During the following six months, little development work was done on R1 either at Digital or CMU because the main focus was on defining a career path for R1 within Digital. But beginning in April 1980, three months were spent at CMU in rewriting the OPS4 version of R1 in OPS5 (Forgy, 1981). Given that the knowledge was already laid out in the OPS4 version, a variety of generalizations emerged and the resulting system, though more capable, had only 500 rules.

By the end of 1980, R1 had 850 rules, most of which were added by people at CMU to provide R1 with additional functionality; the primary focus at Digital during the second half of 1980 was on adding component descriptions to the data base and providing a group of people with the skills to take over the continued development of R1. Most of the work on R1 since early in 1981 has been done by people at Digital. By March 1981, the group at Digital had extended R1 so it could configure VAX-11/750 systems. During the remainder of 1981, most of the group's effort was focused on refining R1's knowledge of how to configure VAX-11/780 and VAX-11/750 systems. In 1982, the focus changed to extending R1 to cover more systems. While some effort was spent in improving R1's performance, substantial effort was spent in extending its scope. By March, a few months before the VAX-11/730 was announced, R1 was able to configure VAX-11/730 systems, and by July, R1 was able to configure PDP-11/23+ systems. At that point, R1's knowledge base consisted of about 2000 rules. The remainder of 1982 and the first few months of 1983 were devoted primarily to refining that knowledge. At that point, a concerted effort was made to extend R1's capabilities so it could configure all the systems sold by Digital in significant volume. When that task was finished in November 1983, R1 had about 3300 rules and its data base contained about 5500 component descriptions. While a significant amount of time will continue to be devoted to extending R1's capabilities to cover new systems as they are announced, effort will also be spent in continuing to deepen R1's expertise in the configuration domain.

As Digital has become more dependent on R1, it has become increasingly important that R1 be highly reliable. Thus substantial attention has been paid to the question of how to combine the demands of reliability with those of continuous development. Early on, little attention was paid to formalizing the developmental process; as problems were reported,

	NUMBER OF RULES	AVERAGE RULES PER SUBTASK	AVERAGE NUMBER OF PARTS ORDERED	AVERAGE RULE FIRINGS	PERCENT OF KNOWLEDGE FREQUENTLY USED	NUMBER OF PARTS IN THE DATABASE
THE INITIAL R1	777	7 6	88	1056	44%	420
THE CURRENT R1	3303	10 3	78	1064	47%	5481
VAX-11/785	2883	9.8	163	2654	24%	3398
VAX-11/780	2883	9 8	171	1925	31%	3398
VAX-11/750	2801	9 7	111	1300	29%	2915
VAX-11/730	2810	9 7	85	1141	29%	2489
VAX-11/725	2788	9 7	34	622	8%	1981
MICROVAX-1	1516	7 3	34	546	18%	1490
MICRO-PDP11	1516	7 3	44	546	18%	1828
PDP-11/23+	1516	7 3	49	608	20%	1894
PDP-11/24	2786	9.7	43	567	13%	1763
PDP-11/44	2786	9 7	43	733	15%	1764

A comparison of the initial and current versions of R1.

Figure 2.

individuals would collect the needed knowledge, add it to the system, and depending on the press of other problems, do more or less testing to determine that the overall capability of the system had not worsened. As time passed, the developmental process acquired substantially more structure. Planned release dates are now preceded by extensive testing of the system.

The article describing the initial version of R1 (McDermott, 1982) provides some insight into the nature of R1's knowledge by presenting a variety of measurements. Figure 2 compares the measurements from the initial version of R1 with corresponding measurements from the current version. Since a significant amount of the knowledge in the current version is specific to just a subset of the system types it can configure, Figure 2 provides the measurements for system-specific configurers as well as for the union of those configurers. Until recently, instead of a single version of R1 that could configure all system types, there was a version of R1 for each system type. Each of these versions consisted of a set of from 50 to 100 rules specific to a system type and two much larger sets of rules; it shared one of these rule sets with all of the other system types and the other with the system types having the same primary bus. About 300 of the shared rules were themselves specific to just one of the system types; each of these rules was included with the shared rules because it was relevant to a shared subtask.

R1's rules are grouped together on the basis of the subtask to which they are relevant; the "number of rules" column displays the total number of rules available to R1 in performing the configuration task, and the "average number of rules per subtask" column displays the mean number of rules in a group. The 3303 rules the current R1 has is the union of the rules of each system-specific configurer; the 10.3 rules per subtask is the union of the groups of rules the system-specific configurers bring to bear on a particular task. The "average number of parts ordered" column displays the number of components R1 has to configure. This number is significantly larger than the number of components listed on a purchase order since those line items actually refer to bundles of configurable components.

The numbers in the "average rule firings" and "percent of knowledge frequently used" columns are based on small sets of runs. For the initial R1, the numbers came from running R1 on 20 typical orders. For the current R1, the numbers came from running each system-specific version of R1 on about 20 orders of comparable complexity. The "average rule firings" column shows that substantially more is done in configuring a VAX-11/780 order now than was done initially; almost twice as many rules are applied. Two factors contribute to this increase. The configuration task has been enlarged by definition (*i.e.* there is now more to do), and second, there has been an increase in the average number of components per order.[2]

The "percent of knowledge frequently used" column shows what percentage of the rules are used at least once in at least one of the sample runs. Thus for the initial R1, 44% of the 777 rules were applied at least once over the 20 sample runs, and for the current R1, 47% of the 3303 rules were applied at least once over the approximately 200 sample runs. The fact that a substantial fraction of R1's knowledge is used only rarely is, of course, just what we would expect of a knowledge-based system. But the percentages for the system-specific versions are somewhat misleading. We would expect the percentage for each version to be lower than the overall percentage because each was run on only about 20 orders. However, because each version has knowledge that is not relevant to its tasks, the percentages for the versions are lower than they otherwise would be. The percentages for the VAX-11/780, the VAX-11/750, and the VAX-11/730 are the most accurate, but even they are too low by several percentage points. Since the nature of the knowledge used by

[2]On the average, 1.67 VAX-11/780 cpu minutes are required to configure an order.

each version is quite similar, it is likely that the percentage of the knowledge frequently used by each is pretty much the same—somewhere between 35% and 40% .

About 65% of the 2526 rules added to R1 since 1980 extend R1's general configuration capabilities; only about 35% of the rules are specific to a single system type. Of the 65% at least 15% were added to correct or refine knowledge of how to perform some subtask. This lower bound is suggested by the fact that the "average number of rules per subtask" increased by 30% during the past four years (*i.e.*, about 230 rules were added to the groups of rules applicable to the subtasks the initial R1 knew how to perform); adding a rule to the group applicable to some subtask is almost invariably done to correct or refine the knowledge of how to perform that subtask. The 15% is a lower bound because as the knowledge required to perform some subtask grows, it may become evident that what was viewed as a single subtask can be viewed as two or more simpler subtasks; what we do not know is how much the average number of rules per subtask would have grown if this subtask splitting had never occurred.

The Kinds of Changes R1 Has Undergone

As it turned out, the task of developing R1 had just begun when it was first put into use. In this section, we attempt to give a flavor of the kinds of changes that have been made to R1 over the past four years by examining a few examples in some detail. Our primary purpose in examining the growth of R1's knowledge is to better understand what is involved in adding knowledge to such a system. We can identify four reasons why knowledge was added to R1:

- To make minor refinements (adding knowledge to improve R1's performance on an existing subtask);
- To make major refinements (adding the knowledge required for R1 to perform a new subtask);
- To extend the definition of the configuration task in significant ways.

Ordinarily when people talk about why knowledge is added to an expert system, they seem to have the first reason in mind. As we have seen, of the more than 2500 rules added to R1 during the past four years, the data in Figure 2 suggest that more than 10% have been added to make minor refinements, fewer than 40% have been added to make major refinements, at least 35% have been added to provide functionality needed to deal with new system types, and perhaps as many as 15% have been added to extend the definition of the task in significant ways.

Minor Refinements. A knowledge addition of the first type is required when R1 cannot perform some subtask that it was thought to be able to perform. For example, over the years R1 has made several errors involving the placement of backplanes in boxes. One instance of such an error has to do with a backplane's location. In one variety of a 24 slot box, because of power considerations, a backplane is not permitted to cover slot 10. R1 knew that if it covered slot 10 when placing a backplane, it needed to move that backplane toward the back of the box so the backplane's front edge would be in slot 11. R1's knowledge was incomplete because it did not know it had to move any previously placed backplane from the front of the box toward the middle so that its back edge would be in slot 9. This backplane has to be moved toward the middle because leaving a larger space between the two backplanes would mean the standard cable used to connect backplanes could not be used (since it is not long enough). Fixing R1 was a straightforward task, but it required a certain amount of creativity (*i.e.*, it was not just a matter of "adding some more domain knowledge.") What R1 lacked was any notion of "deliberately vacant space." In order to provide rules that could recognize situations in which blank space was inappropriately positioned, R1 had to have the concept of blank space and an understanding of how to make a note that a particular space had been left blank on purpose. Given this, it was straightforward to add a few rules that recognized when some piece of blank space was inappropriately located and swap it with a backplane.

Major Refinements. A knowledge addition that results in a major refinement to R1 can be made in two kinds of situations: when R1 does not have any knowledge about how to perform some subtask, and when its knowledge of how to perform some subtask becomes so tangled that ways need to be found of representing the knowledge more generally. Brief examples of both situations are presented below; in the following section we provide a more lengthy analysis of one attempt to rewrite a set of rules, initiated almost purely to increase generality and understandability.

Most of the modules R1 configures on a UNIBUS consist of one or more boards that plug into backplanes which go in boxes. If multiple boards are required, they are usually placed next to each other in the same backplane. A situation unfamiliar to R1 arose when a module was designed with boards on two buses. Its first board was to be configured in an SPC backplane while the three remaining boards were to be configured in a special backplane that had to be located in the same box as the first board, but not in the same backplane. One way of extending R1 to handle this new component would have been to use a look-ahead strategy; R1 would have checked for space, power, and cabling constraints on the special backplane before configuring the first board. An alternative would have been a simple backtracking strategy. The approach R1 actually took involved a combination of both look-ahead and backtracking. R1 applies the same rules it uses for other modules to configure the first board; a few special rules then try to foresee abstract constraint violations involving the rest of the boards. If a problem is found, the first board is unconfigured. If no constraints are violated, power and space are reserved for the remaining boards.

Early in R1's history, only two types of panels needed to be considered. A few rules were sufficient to guard against the possibility of trying to configure two panels in the same

space in a cabinet. Templates were used to describe panel placement possibilities; the rules recognized when some particular space was already occupied and avoided that space. As Digital introduced new products the situation became increasingly complicated until five different types of panels as well as disk drives and boxes could potentially occupy the same space with differing degrees of overlap. Because the original approach required all possible conflicts to be enumerated, it became increasingly unwieldy as the problem grew in complexity. The new solution involved redesigning the templates so the information they contained could be manipulated by a small number of more general rules and by making minor changes to the action parts of about 60 already existing rules that dealt with cabinet space decisions. This strategy worked well for about a year until Digital redesigned its cabinets to comply with new FCC regulations. At that point, the templates became too unwieldy because of the sheer number of possible individual locations; since the redesign also eliminated most of the irregularities of the previous problem, it became possible to simplify the templates and keep track of potential conflicts with a few very general rules.

New System Types. Providing R1 with the functionality it needed to deal with new system types has constituted a significant portion of the development effort. Since major configuration differences exist among the various buses supported by different CPU types, it was not clear initially how much configuration knowledge is common across system types. When a VAX-11/750 configurer was developed, the VAX-11/780 configurer was used as a model, but the knowledge bases were initially completely separate. Once the VAX-11/750 configurer had enough knowledge to be useful, it was merged with the VAX-11/780 configurer. On the other hand, the VAX-11/730 configurer was integrated, from the beginning of its development, with the older R1; the new version was developed by creating a small knowledge base (consisting of about 100 rules) specific to the VAX-11/730, adding some rules specific to the VAX-11/730 to the common knowledge base, and generalizing several of the rules in that common knowledge base. This approach worked well for the VAX-11/730, but when we turned our attention to the PDP-11/23+, we reverted to the approach we had used for the VAX-11/750. Several factors were involved in this decision. R1, up to this point, knew only of VAX-11 systems, which are UNIBUS and MASSBUS based, while the PDP-11/23+ is based on the LSI22 bus. The rules for configuring these buses have little in common. Moreover, the PDP-11/23+ supports a variety of operating systems, requires a completely different paneling structure, and assumes different power and capacity characteristics for its boxes and backplanes. Since the PDP-11/23+ is quite dissimilar to the VAX-11 systems, a separate version of R1 was developed for this task. Each of the subsequent system configurers was integrated with either the VAX-11 or the PDP-11/23+ system (depending on whether it had a UNIBUS or a LSI22 bus). Recently, it was decided that in a production environment, it would be advantageous to have one single system; by April 1984, all of the system configurers had been merged. Future system type additions will be part of this single version of R1 from the beginning of their development.

Adding the knowledge required to deal with new system types is non-trivial even when the new type is quite similar to types R1 already knows how to configure. Part of the effort of extending R1's configuration capabilities to cover a new type is due simply to the added amount of knowledge. For each of the types, we have had to add a great deal of data to the data base as well as make extensive rule changes and additions. Many of the decisions involve how to represent the new knowledge in the rules, but new data base representations are sometimes also required. The full extent of the effort varies, depending on the degree of similarity between the added system type and the types R1 can already configure. When there is a high degree of similarity, the form in which the existing knowledge is represented provides substantial guidance for how to represent the new knowledge. When the new system is quite dissimilar, substantial amounts of design are required.

Extending the Task Definition. As R1's role in Digital's manufacturing process has evolved, knowledge has been added to R1 that extends the definition of its task. For example, R1 was extended in January of 1983 to handle "multiple-CPU" orders. R1 was originally designed to deal with orders containing a single CPU. But multiple-CPU orders have become increasingly common, especially with the advent of smaller system types where multiple identical systems and/or several different systems on the same order are the norm. Part of the challenge of extending the definition of R1's task involves finding a way to realize some new capability that does not require extensive modifications to R1. In this case, we avoided the temptation of trying to modify R1 to configure multiple, loosely coupled systems simultaneously. Instead, a few new rules (originally about 10) were written to group the components into individual systems; each system was then configured in turn. Changes had to be made to 5 existing rules that determine what to configure and what order information to save; a few external initialization and output routines also had to be modified. The hard part was determining how R1's task definition could be extended most simply.

A substantial change to R1 in July of 1982 modified it to deal with a different categorization scheme for components. The component descriptions had been developed exclusively for R1 and were tailored to the configuration task. As Digital developed other knowledge-based systems for other purposes, it became desirable to have a common data base, where the components were categorized in a less ad hoc fashion. Before R1 could use the new descriptions, nearly all of its rules (about 2000 at the time) had to be changed, and for several hundred of these rules, the task of reformulation took considerable thought.

While the difficulty of making changes of any of the four types we have just described is highly dependent on the na-

ture and scope of the knowledge that needs to be added, it also appears to be dependent on the amount of knowledge the system already has. In the early days, when R1 was small, people who joined the project were able, reasonably quickly, to acquire enough of an understanding of the configuration task and of R1's approach to it to become competent developers. But now that R1 has grown substantially, its sheer magnitude seems to serve as a barrier to the would-be developer. It takes much longer now for someone who joins the group to gain an adequate understanding of how R1 does configuration.

A Change over the Years

To provide another view of R1's development, we have analyzed the changes in R1's knowledge for two closely related tasks. One of the tasks involves deciding what backplane should hold the next set of modules. The other is a subtask that may or may not be performed depending on what the backplane selection possibilities are. The decision of what backplane to configure next is constrained by the pinning type of the modules, the space and power available for them, the current length of the bus and its loading, and the number and mix of backplanes that have been ordered. A good backplane choice is one that minimizes the number of additional components that have to be added, while satisfying all the constraints. The subtask is performed if the pinning type of the next module to be configured is SPC. In this case, two different sized backplanes could be used, so R1 must do some analysis of the implications of selecting each. Figure 3 shows how R1's knowledge of these tasks has developed; the development can be viewed as a series of minor refinements, followed by a major refinement.

In December 1980, R1's knowledge of how to perform the two tasks consisted of 36 rules, 23 rules for the selection task and 13 rules for the subtask. In October 1983, R1's knowledge consisted of 73 rules, 54 for the selection task and 19 for the subtask. During the intervening three years, 40 rules were added, 3 rules were eliminated, and 11 rules were changed. This alteration is consistent with the knowledge-based approach, where the initial instinct is to solve a problem by adding more knowledge. It suggests that the rules eventually formulated are for the most part adequate, but that it takes a long time to collect the relevant knowledge. The fact that only 11 rules were changed may be a little misleading since 27 of the added rules were special cases of existing rules, implying that the conditional part of many of the unchanged rules were inadequately discriminating. Of the rules that were changed, the changes were almost all in the conditional part and were in the direction of making the rules increasingly discriminating.

In October 1983, one of the people working on R1 observed that if R1 were given more knowledge of how to assess the likely implications of various decisions, it would need to backtrack even less often. In the course of reworking this capability, the number of rules remained constant, but the level of expertise improved dramatically. Of the rules in the October version, 31 were eliminated and, coincidentally, 31 were added; of the remaining 42 rules, 32 were changed. Again, this alteration is what we might expect of a situation in which a capability is being substantially extended. When the knowledge is all laid out and it is clearer what other pieces of knowledge are relevant to the task, it becomes more obvious how to represent the knowledge cleanly. In this case, the biggest change was the elimination of condition elements. This happened because it became clear that the rules were too constraining; that is, the rules had typically been added to deal with a particular error, and so the October version had a small set of overly general rules (from the initial version) and several more overly specific rules. Seeing all the knowledge laid out made it possible to hit the right level of specificity.

	12/15/80	10/15/80	12/15/80
TOTAL RULES	36	73	73
Rules Added	40	31	
Rules Deleted	3	31	
Rules Changed	11	32	
Condition Elements Added	11	8	
Condition Elements Deleted	1	72	
Condition Elements Changed	15	40	
Action Elements Added	1	5	
Action Elements Deleted	0	8	
Action Elements Changed	1	13	

Two sample subtasks

Figure 3.

Conclusions about Growth

The following conclusions purport to provide guidance to the developers of any knowledge-based application system. We are of course not at all sure what aspects, if any, of the experience with R1 at Digital will turn out to be typical. It seems reasonable to believe, however, since R1's task is knowledge-intensive, that the experience with R1 relating to the rate at which it has acquired knowledge and the difficulty of adding that knowledge will at least have relevance to other attempts to put knowledge-based systems to work on real tasks.

Even though the experts claimed in 1979 that R1 had most of the knowledge it needed, a great deal of knowledge has been added to R1 over the past four years. There is no more reason to believe now than there was then that R1 has all of the knowledge relevant to its configuration task. This, coupled with that fact that R1 deals with an ever-changing domain, implies its development will never be finished. Thus users of systems like R1 will have to be emotionally prepared to interact with a less than perfect program. They will have to be as forgiving of ignorance in these expert systems as they are of ignorance in humans who are ever becoming more

expert.

Though much of R1's knowledge was added to correct or complement existing knowledge, a significant part of the additions came as a result of R1 having to have the knowledge to perform new tasks. Some of these were the result of Digital introducing new computer system types and the rest resulted from the users' observations that things would be better if R1 could do one more thing. We believe all expert systems will be hounded to continue to grow for both of these reasons. Tasks that expert systems are good for are just those whose objects change significantly over time. Moreover, in such tasks no clear boundaries delimit what should and should not be within the province of the expert. Thus, whenever an expert system finds itself on a boundry, its public encourages it to extend the boundary.

Situations arise in which the task of adding a piece of knowledge is extremely straightforward because the new knowledge needs to be represented and used in virtually the same way as the system's existing knowledge. But, for the most part, adding a piece of knowledge involves some amount of creativity. In domains other than configuration (or at least in diagnostic as opposed to constructive tasks) domain knowledge appears to be substantially more regular and can be added routinely. Significant, but as yet undiscovered, regularities in configuration knowledge may exist that will someday allow it to be added more easily. But for now, it is important to at least be open to the possibility that a knowledge-based system will forever have to be surrounded by people who know how to do development. They will be called upon to be innovative and adaptable. Although it may be the case that adding knowledge incrementally is easier than rewriting or modifying a traditional program, by no means can this task be done without substantial amounts of problem solving.

It was clear before R1 was a year old that the incremental addition of knowledge resulted in a system with a significant amount of redundancy and a penchant for *ad hoc*ery. To the extent that adding knowledge to the system involves human intervention, this general lack of cleanliness and conciseness provides an obstacle to the system's further development. Few expert systems are likely to be redeveloped (as R1 was in 1980, but not since). However, we suspect that from time to time, some part of every expert system's knowledge will become so convoluted that its developers will take the time to re-represent that knowledge.

R1's Performance

Before R1 began to be used, each system Digital received an order for was configured by a technical editor, typically on the day before the system was to be assembled and tested. The technical editor examined each order to determine whether configuration constraints required additional or different components and then specified some of the relationships among the components on the order. Though the task was performed at a fairly high level of abstraction, it seldom took fewer than 5 or 10 minutes to configure an order, and complex orders took substantially more time. When R1 began to be used, it essentially became a technical editor. But since it was not clear initially how well R1 was going to do as a technical editor, some of the people who had been technical editors stayed to watch over R1. In effect, they became R1's mentors. Every order configured by R1 has been examined, more or less closely, by a mentor and if the mentor believed the configuration was lacking in any respect, he or she reported the problem to the R1 development group.

Although R1 is an expert system in the sense that the body of knowledge it uses to perform the configuration task is acquired by human experts over a period of years, its task is different from the task that used to be performed by the technical editors because R1 configures systems at a significantly greater level of detail than they did. Because its task is more extensive, it is hard to answer the question: Does R1 do as well at the technical editing task as human experts do? The task R1 actually performs is the old technical editing task plus part of the task performed by the technician who physically assembles the system (since the technician has to descend to R1's level of detail to do his job). But the technician's situation is different from R1's in that the technician has the physical components that need to be assembled and tested in front of him and can discover when components are missing or misconfigured in more direct ways than are available to R1. Thus we have not tried, in this article, to compare R1's performance with that of the human experts. The closest we come to examining that relationship is with the bogus problems category. A bogus problem is one that a human expert reports as an R1 error, but that on further examination turns out to have been a failure on the part of the expert to appreciate correctness.

The data presented in this part of the article leave something to be desired; part of the problem is that it was not clear, at any point during the past four years, how frequently R1's performance needed to be sampled. Since knowledge-based systems continue to be developed incrementally as they are used, it was obvious that collecting performance data would be an integral part of using the system. It was also clear that the more data that were collected, the better we would understand the extent to which R1's knowledge was incomplete. But all that is really required to drive the developmental process is enough data to give the people collecting and encoding R1's knowledge plenty to do. Since finding inadequacies in R1's knowledge has never been very hard, more attention was given to the task of extending and refining R1 than to the data collection task. As a result, there are a few periods, in two cases extending for months, in which the data we have are incomplete. For the most part, however, we have some information about how well R1 performed on each order it configured.

Even if we had information about each order R1 configured, our data would still be unsatisfactory because our understanding of how to collect the relevant data has grown slowly. Since people who have the responsibility of review-

ing each of R1's configurations have little understanding of how R1 does what it does and where and how it can err, they can only report error manifestations. Devising a process that makes it fairly straightforward to link manifestations to causes (so, for example, the number of instances of each error type can be determined) took some time. Initially the process used paper and pencil. A second issue, then, was how to design a program that could assist with the data collection task. Because it took time to devise such a program (a lot of time since it had low priority), a significant part of our task has been to reconstruct, from incomplete descriptions of error manifestations, what the actual errors were. We feel relatively confident in the overall results, but are sure a number of minor inaccuracies exist.

Before presenting the performance data, we need to discuss briefly how "percentage of totally correct orders" came to be accepted early as the metric for measuring R1's performance. The problem with this metric, of course, is that it does not discriminate between terrible performance (gargantuan errors) and near perfect performance (tiny, almost insignificant errors). In retrospect, it is clear that having some idea of the seriousness of each error would be helpful in evaluating R1. But when R1 first started to be used, it was with the expectation that there were only a few things it did not yet know, and the only question in people's minds was how many weeks it was going to take before R1 knew everything. Within that context, it is not at all surprising that the all or nothing metric was selected; anything else would have seemed too fine-grained.

Some Performance Data

Figure 4 provides a detailed account of R1's performance over the past four years. The information is presented by quarter, beginning in January 1980 and ending in December 1983. Three major problem categories exist: rule problems, data base problems, and other problems. For rule and for data base problems, as well as for total problems, the percentage of orders containing that type of error is given. Within each category, information is provided about one or more subcategories. For each subcategory, the number of problem instances as well as the number of distinct problems is reported. The total problem instances percent gives a sense of R1's usefulness. However, since most errors R1 now makes are minor, its output, even if there are problems, can usually be used, though sometimes only after a bit of editing. The distinct problems percent in the parts and rules subtotal gives a sense of R1's competence; this measure shows the number of distinct errors R1 has made due either to missing or incorrect configuration knowledge or to missing or incorrect component descriptions. Few, we think, would want to claim that R1 was a competent configurer during its first year of use; but for the past two years, its lack of knowledge has been well within the bounds of respectability. The number of problem instances divided by number of distinct problems gives an indication of how many times a problem occurs before it is fixed.

The most significant improvement in R1 has come in the percent of problems attributable to missing or incorrect rules. While missing or incorrect domain knowledge has never been the most significant source of problems, it is now the case that fewer than one in a thousand orders is misconfigured because of rule problems. One might ask (though we hope only in jest) how after four years R1 can have any missing or incorrect domain knowledge. There are at least two answers. First, even though R1 has configured more than 80,000 orders, it has seen only a small fraction of the situations it could possibly encounter. Second, new products are sometimes announced before R1 acquires all the knowledge it needs to be able to configure those new products correctly.

Problems with parts have been much more troublesome. Incorrect part descriptions have never been much of a problem, but missing part descriptions have been a significant problem during all four years. During the first two years R1 was used, the reason it was sometimes given systems to configure containing components not described in its data base had mostly to do with the fact that the people responsible for adding part descriptions to the data base were not the right people. It was assumed initially that the component descriptions could be created by people who knew a lot about the components, but knew little about how R1 would use the descriptions. As it turned out, creating useful descriptions is not all that straightforward. It often is not clear what "configuration level" means, not clear what attributes are required, and not clear what knowledge to put in the rules and what in the data base. In order to know what information a description should contain, it is necessary to know how the information is going to be used. In order to know how the information is going to be used, it is necessary to know something about the component. After trying various strategies for making the middle-men more productive, the responsibility for creating descriptions was taken over by the people who encode the configuration knowledge in rules.

This change would have solved the missing part descriptions problem were it not that at about that time, the number of orders R1 was configuring per quarter began to increase substantially. As a consequence, the number of different parts ordered grew significantly. Since R1 has descriptions of only 5,500 of the more than 100,000 parts that could appear on an order, and since the rate at which the as yet undescribed parts appear on orders is very low, the development group adopted the strategy, for low volume parts, of waiting until the part shows up on an order before adding its description to the data base. This policy is less cavalier than it may seem since when one of these low volume parts does show up on an order, it usually turns out to be a part that is not itself configured (*e.g.*, software or an accessory). Thus although any configuration mentioning a part R1 does not know about is counted as a problem, most of the time those configurations can be used without modification.

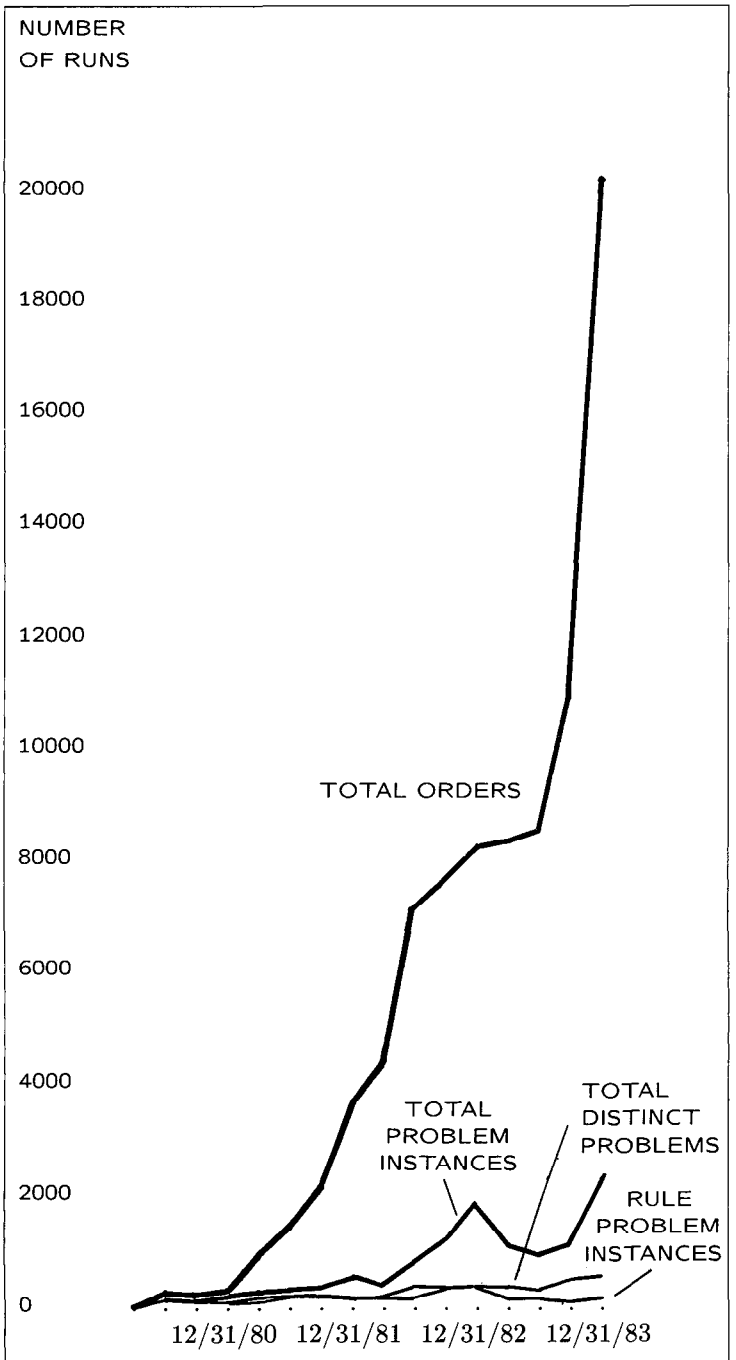
	1st Qtr 1980	2nd Qtr 1980	3rd Qtr 1980	4th Qtr 1980	1st Qtr 1981	2nd Qtr 1981	3rd Qtr 1981	4th Qtr 1981	1st Qtr 1982	2nd Qtr 1982	3rd Qtr 1982	4th Qtr 1982	1st Qtr 1983	2nd Qtr 1983	3rd Qtr 1983	4th Qtr 1983
Incorrect Rules																
Problem Instances	6	21	12	26	55	–	–	171	123	313	243	342	100	58	13	80
Distinct Problems	6	7	8	13	18	–	–	30	25	41	38	37	16	14	6	19
Total Orders	54	194	133	210	824	–	–	3605	4283	7100	7503	8110	8192	8427	10775	20241
Problem Instances Percent	11.1%	10.8%	9.0%	12.4%	6.7%	–	–	4.7%	2.9%	4.4%	3.2%	4.2%	1.2%	0.7%	0.1%	0.4%
Distinct Problems Percent	11.1%	3.6%	6.0%	6.2%	2.2%	–	–	0.8%	0.6%	0.6%	0.5%	0.5%	0.2%	0.2%	0.1%	0.1%
Missing Part Descriptions																
Problem instances	6	51	16	22	33	–	–	136	78	214	629	997	697	535	906	1962
Distinct problems	6	44	16	19	31	–	–	43	42	40	105	157	162	141	368	472
Orders with errors	4	21	7	12	19	–	–	104	65	166	439	601	535	464	920	1275
Incorrect Part Descriptions																
Problem instances	1	9	17	15	21	–	–	74	31	86	94	86	23	19	10	28
Distinct problems	1	3	3	6	12	–	–	12	11	16	9	19	8	6	5	9
Parts Subtotal																
Problem instances	7	60	33	37	54	–	–	210	109	300	723	1083	720	554	916	1990
Distinct problems	7	47	19	25	43	–	–	55	53	56	114	176	170	147	373	481
Total orders	54	194	133	210	824	–	–	3605	4283	7100	7503	8110	8192	8247	10775	20241
Problem instances percent	13.0%	30.9%	24.8%	17.6%	6.6%	–	–	5.8%	2.5%	4.2%	9.6%	13.4%	8.8%	6.6%	8.5%	9.8%
Distinct problems percent	13.0%	24.2%	14.3%	11.9%	5.2%	–	–	1.5%	1.2%	0.8%	1.5%	2.2%	2.1%	1.7%	3.5%	2.4%
Parts and rules Subtotal																
Problem instances	13	81	45	63	109	116	190	381	232	613	966	1425	820	612	939	2098
Distinct problems	13	54	27	38	61	–	–	85	78	97	152	213	186	161	384	509
Total orders	54	194	133	210	824	1304	2040	3605	4283	7100	7503	8110	8192	8427	10775	20241
Problem instances percent	24.1%	41.7%	33.8%	30.0%	13.3%	8.9%	9.3%	10.5%	5.4%	8.6%	12.8%	17.6%	10.0%	7.3%	8.7%	10.4%
Distinct problems percent	24.1%	27.8%	20.3%	18.1%	7 4%	–	–	2.3%	1.8%	1.4%	2.0%	2.7%	2.3%	1.9%	3.6	2.5%
Operational Problems																
Problem instances	2	45	16	56	77	115	67	12	7	2	41	0	11	1	0	0
Controversial Issues																
Problem instances	0	1	3	3	9	11	47	80	21	33	19	26	50	46	19	25
Distinct problems	0	1	2	1	3	–	–	4	3	8	9	5	8	7	4	4
Desired Enhancements																
Problem instances	–	–	–	–	–	–	–	38	6	19	11	57	86	155	25	27
Distinct problems	–	–	–	–	–	–	–	3	2	6	1	2	5	7	3	4
Bogus Problems																
Problem instances	0	3	1	7	27	16	31	15	15	36	76	62	44	23	33	43
Total Problem Reports																
Problem instances	15	130	65	129	222	258	335	526	281	703	1113	1570	1011	837	1016	2193
Distinct problems	15	103	46	102	168			119	90	149	279	282	254	199	424	560
Total orders	54	194	133	210	824	1304	2040	3605	4283	7100	7503	8110	8192	8427	10775	20241
Problem instances percent	27.8%	67.0%	48.9%	61.4%	26.9%	19.0%	16.4%	14.6%	6.6%	9.9%	14.8%	19.3%	12.3%	9.9%	9.4%	10.8%
Distinct problems percent	27.8%	53.1%	34.6%	48.6%	20.4%	–	–	3.3%	2.1%	2.1%	3.7%	3.5%	3.1%	2.4%	3.9%	2.8%

R1's Performance by Quarter

Figure 4.

The problems not really under the control of R1's developers—operational problems, controversial issues, desired enhancements, and bogus problems—have always been a significant part of the problems reported. During the first year and a half, a very large fraction of the problems were operational; a number of factors, each by itself not very significant, conspired to separate R1 from its user community. During 1983, the number of bogus problem reports grew to become a highly encouraging (from R1's point of view) fraction of the total problem reports; from July through September, the number of bogus problem reports was actually double the number of rule problem instances, and during the other three quarters the number of bogus problem reports was about half the number of rule problem instances.

Figure 5 presents some of the information from Figure 4 in graphical form. The relationships among "total orders", "total problem instances", "total distinct problems" and "rule problem instances" are depicted. The "total orders" measure provides a context within which the error measures can be understood. The "total problem instances" measure provides a lower bound on R1's usefulness. The area under that curve indicates the number of orders for which R1's output was possibly not useful; however, as we have seen, in most cases the output could be used, though sometimes only after being modified. The "total distinct problems" measure provides a lower bound on R1's competence. The area under that curve indicates the number of different kinds of situations R1 did not deal effectively with. The "rule problem instances" measure indicates the extent to which R1's failures were due to its ignorance of the domain.



R1's Performance by Quarter (in Graphical Form)

Figure 5.

Conclusions about Performance

As in the previous section where some conclusions about growth were presented, the following conclusions purport to provide guidance to the developers of any knowledge-based application system. Since the conclusions we offer here are not very startling, it is quite likely that they have some general validity. All they really contain is the notion that when AI tools confront real tasks, the world is not going to obediently conform to all of the hopes of the tool maker. The real world treats AI tools with the same disrespect with which it treats all other tools and thus a great deal of the effort of bringing AI systems into regular use on real tasks involves doing things that do not have any special relationship to AI. What undoubtedly makes matters worse for AI tools is that the problems they are used to solve are ordinarily more open than the problems traditional software tools typically address.

In the previous section we argued that an expert system will never have all the knowledge it needs. Thus it will always make mistakes, and it is important for both the developers and the users to expect them. R1's performance data suggest something even stronger: To expect anything close to perfection during the first few years a system is being used (especially if the task is significantly more than toy) is probably a very serious mistake. We believe the data also suggest that to keep an expert system from regular use until its knowledge is complete would be a poor idea. It has taken 80,000 orders to uncover some of the inadequacies in R1's configuration knowledge, and the configuration task is continually redefined as new products are introduced. These facts suggest that even if someone had the time and energy to try to create a near perfect system before introducing it into production, many inadequacies would become evident

with regular use.

It would be a mistake to believe the major or even a primary source of error in the performance of an expert system will be due to incorrect or missing domain knowledge. Depending on the number and type of objects the system is intended to deal with, large amounts of effort may be needed to collect and maintain the information about these objects. But even if the nature of the task makes data collection and maintenance relatively unproblematic, a variety of other sources of error may spring up as the system begins to be used. As we just mentioned, there is nothing magic about knowledge-based systems that allows them to avoid the problems other software systems have to face. Indeed, the fact that they continue to be developed while they are being used undoubtedly intensifies those problems. The relative seriousness of the various problems that confronted R1 would surely have been better appreciated if R1 had had a sophisticated problem reporting mechanism from the beginning.

If one looks at R1's performance over the first two years of its use and tries to imagine R1 being used in a situation where it was being asked to configure thousands of orders a month, it seems clear that its use would have been discontinued. This judgment is perhaps overly harsh since, as mentioned above, a significant portion of the configurations with errors could be used with only minor modifications. In any event, using R1 in a high volume environment would have made its initial nuturing substantially more difficult. R1 was used instead in an environment in which the initial demands on it were of the order of a few tens of orders per week for the first year. This small volume made it possible for people to jump in whenever R1 failed and to avoid depending too much on a system that at the time was far from being an expert in the domain.

Conclusion

One of our purposes in giving these glimpses of R1's developmental and performance histories is to provide some evidence for evaluating the claims that have been made about expert systems. Expert systems supposedly are easy to develop incrementally and, at some point, become as good as human experts. R1 lends some credence to both of these claims. While substantial effort has been required to develop R1, the approach taken has made it possible over a four year period to increase R1's knowledge substantially without starting over; this lends support to the first claim. The fact that human experts erroneously conclude that R1 has misconfigured systems about as frequently as R1 actually misconfigures systems lends some support to the second claim.

References

Forgy, C. L. (1979) OPS user's manual. Technical Report, Carnegie-Mellon University, Department of Computer Science.

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

Gupta, A and C. L. Forgy (1983) Measurements on Production Systems. Technical Report, Carnegie-Mellon University, Department of Computer Science.

McDermott, J. (1980) R1: An expert in the computer systems domain. *In Proceedings of AAAI-80, National Conference on Artificial Intelligence,* Stanford, California, 269-271

McDermott, J. (1981) R1's formative years. *AI Magazine*, Vol. 2., No. 2, 21-29.

McDermott, J. (1982) R1: A rule-based configurer of computer systems. *Artificial Intelligence* 19(1), 39–88 Also available as a CMU, CSD technical report.