Extending a Knowledge-Based System to Deal with Ad Hoc Constraints

John McDermott and Barbara Steele
Carnegie Mellon University
Pittsburgh, Pennsylvania

Abstract. R1 is a rule based program used by Digital Equipment Corporation's manufacturing organization fc configuring VAX 11 systems. Since its inception. R1 has had the capability of producing a functional system by fleshing out an order consisting only of a CPU. some primary memory, and some devices; until recently, however, it was not capable of accepting as part of its input a set of ad hoc (customer specific) constraints Left to itself, R1 was capable of producing reasonable configurations. But it was incapable of modifying its decisions on the basis of information that others could provide This paper describes how rules were added to R1 to take advantage of such information R1 can now accept as input commands that specify how particular components are to be configured Whenever one of the commands becomes relevant, these rules take control, extend the configuration in the direction indicated by the command, and then step aside, allowing Rt's ordinary-case configuration rules to regain control.[1]

1. Background

In December, (978. we began v/ork on R1, a rule based program that configures VAX 11 computer systems [McDermott 80a, McDermott 80bl. Given a set of components (a customer's order). R1 produces diagrams specifying the spatial relationships among those components. If. while organizing the components, it finds that additional components are needed to construct a functional system, it adds those components. Since January, 1980, R1 has been used by Digital Equipment Corporation's manufacturing organization to configure almost all VAX 11 systems shipped Digital currently has a group of 15 people responsible for maintaining R1 and extending its capabilities.

The development of. R1 and XSEL is being supported by Digital Equipment Corporation The research that led to the development of OPS5. the language in which RI and XSEL are written, was sponsored by the Defense Advanced Research Projects Agency (DOD). ARPA Order No 3597, and monitored by the Air Force Avionics Laboratory under Contract F33615 78 C 1151 The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Digital Equipment Corporation, the Defense Advanced Research Protects Agency, or the US Government VAX 11 and Unibus are trademarks of Digital Equipment Corporation

R1 is implemented in OPS5, a general purpose, rule-based language [Forgy 77, Forgy 80]. OPS5 provides a *rule memory,* a global *working memory,* and an *interpreter* that tests the rules to determine which ones are satisfied by a set of the descriptions in working memory. A rule.is an IF THEN statement consisting of a set of conditions (patterns that can be matched by the descriptions in working memory) and a set of actions that modify working memory On each cycle, the interpreter selects one of the satisfied rules and applies it. Since applying a rule results in changes to working memory, different subsets of rules are satisfied on successive cycles. R1s knowledge of how to configure VAX 11 systems consists of about 850 rules; each of these rules encapsulates one set of constraints on the way in which components can be associated. Initially working memory contains just a list of the names of the components ordered. After a description of each of these components is retrieved from a data base. RI constructs a configuration by associating components in ways that satisfy the constraints held in the rules. R1 is recognition driven; that is, each of its rules recognizes a different situation in which the creation or extension of a partial configuration is called for R1 does almost no backtracking because its knowledge is sufficient to lead it unerringly from the set of components ordered to an acceptable configuration.

XSEL, a system that will complement R1, is currently under development at CMU [McDermott 81]. XSEL is a computer salespersons assistant; its function is to interact with a salesperson (or customer) to obtain the information required in order to tailor a system to fit the customer's needs. After obtaining this information from the customer, it selects 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 is then 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 that the particular needs of that customer imply special configuration constraints; in these cases XSEL informs R1 of those constraints.

R1 was not designed*to take customer-specific constraints into account in configuring systems. Therefore, for R1 to use the

information passed to it by XSEL, a major modification to its approach to the configuration task was required The extended task imposes significant new demands:

- Customer-specific constraints must be viewed as refinements of RVs ordinary-case configuration constraints.
- Systems must sometimes be configured in the absence of customer-specific constraints.
- Customer specific constraints can affect most aspects of a system's configuration.

In the absence of information about how a customer intends to use his system, the configuration task may be under-constrained; there may be a number of apparently equally acceptable ways in which the system can be configured. Given some information about how the customer intends to use his system, some configurations become more acceptable than others, and occasionally configurations that appeared unacceptable become acceptable. Thus customer-specific constraints must take precedence over the ordinary-case constraints. The uses a customer intends to make of his system sometimes imply no special constraints Thus customer-specific constraints cannot be presupposed. Finally, the needs of a customer may have implications for almost any aspect of the system configuration. Thus no ordinary-case constraint knowledge is unaffected by customer specific constraints.

Because the task of extending R1 to handle customer-specific constraints has implications for all of RVs knowledge, we considered redesigning R1. But that turned out not to be necessary. Instead, we were able to develop a general ad hoc constraint-handling capability by augmenting R1 with a relatively small number of rules. The general problem we faced is that of transforming rule-based programs that are unable to take advice into programs that can use advice or do without it. The problem divided into two parts. Most of RVs configuration capabilities have gone through a lengthy period of test and refinement; the knowledge it needs for these capabilities is now relatively complete. Recently, however, R1 was augmented with a set of rules that enable it to do the floor layout task; this capability is just beginning to be tested and refined. In the next section of the paper, the ad hoc constraint-handling capability used by R1 for tasks in which it already has expertise will be described. Section 3 will address the issues that arose in providing such a capability for the floor layout task.

2. A Superimposed Ad Hoc Constraint-Handling Capability

RVs expertise in the configuration task has evolved gradually over a period of two years. Its knowledge is now sufficient to enable it to produce an acceptable configuration in the absence of information about the intended uses of a system. Thus our task was to preserve this expertise while at the same time modifying R1 so that it could benefit from the advice of experts who understand the special configuration constraints implied by particular applications.

2.1. Design Considerations

In order to understand the problem of oroviding R1 with the capability of assimilating constraints on the fly, it is necessary to know something about the basic problem-solving method that R1 uses. The method used is a particularly powerful variety of heuristic search called Match [Newell 69). When Match can be used to solve a problem, a path from the initial state to a solution state ran be found without any backtracking. It follows that for Match to be applicable, it must be possible to order the set of relevant decisions in such a way that no decision has to be made until the information sufficient for making it is available. It should be clear that for complex problems (in which much of the information required at any decision point will be available only after other decisions are made) the applicability of Match rests entirely on whether it is possible to specify at each step what information is sufficient for that step. For most of the VAX-11 configuration task, it is possible to provide such a specification.

R1 implements Match by requiring that the conditions of each of its rules specify the conditions under which that rule can be safely applied; that is, no rule can be applied in a particular situation (or at a particular state) unless all the information needed to insure that it will extend the configuration in an acceptable way is present. A typical rule is satisfied if (1) an unconfigured component with a particular set of characteristics is in working memory, and (2) descriptions of partial configurations and/or other unconfigured components with particular characteristics are in working memory. When the rule is applied, it specifies how that unconfigured component is to be associated with one or more of the other components.

Since each of RVs rules implicitly defines exactly the set of situations in which it can be applied, there are no necessary interdependencies among RVs rules. A rule that recognizes a situation as one it can deal with does not care how that situation came to be; it cares only that it has come to be. This suggests that all that is needed to implement an ad hoc constraint handling capability is a set of rules that (1) recognize when various types of customer-specific constraints have become relevant to the current state of the configuration and (2) step in to effect whatever extension or modification is called for. RVs other rules, since they are indifferent to the genealogy of a configuration, will simply treat these externally imposed changes as if they had been initiated by R1. The new rules are, of course, subject to the same requirement as RVs other rules: they have to define the conditions under which they can be safely applied. But this poses no serious problem since the new rules propagate constraints that tailor the configuration to the customer's needs, and thus the only issue is when they should be applied, not

whether they should be applied.

More specifically, 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 hag\ We simply added a set of rules tnat recognize situations in which some type of customer-specific constraint is present together with descriptions of components or partial configurations which indicate that 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 customer-specific constraint. In a sense, these new rules are disassociated from the mainstream configuration task; they stand outside and allow customer-specific demands to step in, change the world to be the way the customer wants it, and then step back out and let R1 continue about its business.

2.2. A Language for Communicating Constraints

To understand the nature of the constraints that XSEL might pass to R1, it is necessary to have a rough idea of the dimensions along which acceptable configurations can vary. There are four dimensions of primary interest.

- The positions of controllers on buses.
- The positions of backplanes in boxes or cabinets and the positions of boxes and panels in cabinets.
- The distribution of devices among controllers
- The types and lengths of cables used to connect pairs of devices

Along any of these dimensions, customer specific needs may dictate specific partial configurations or may simply indicate that the general configuration constraints should be made more or less restrictive. Thus XSEL must be able to tell R1 (1) to configure a set of components in a particular way, and (2) to strengthen or relax particular constraints.

The language XSEL uses to communicate with R1 consists of a small set of commands that give XSEL control over the various aspects of the four dimensions listed above. R1 understands 20 commands. Most of these commands enable XSEL to specify how a particular component is to be associated with other components; the rest enable it to modify various restrictions that R1 ordinarily assumes. When XSEL discovers a customer-specific constraint, it generates a command that tells R1 how to satisfy the constraint. When R1 is given a set of components to configure, any commands generated by XSEL are passed to it as well. Each command becomes a working memory element containing information that identifies the command as relevant to a particular set of components or to a particular restriction.

Each of the commands that specifies how a component is to be associated with other components contains a descriptor that indicates the purpose of the command, and identities by name or by type the component it is concerned with. A variety of other information is contained in a command, all of which has the function of specifying the nature of the relationship between the component and other components. Sometimes this information identifies a second component and describes precisely how the two components are to be associated (e.g., the position of a backplane in a box, the location of a controller on a bus). In other cases, no other components are explicitly identified; the relationship of the component to other components is specified indirectly (e.g., the position of a module relative to otner modules). Each command that describes how R1 is to modify a restriction identifies the restriction to be changed and provides a redefinition of that restriction.

The command that enables XSEL to specify precisely where a Unibus module is to be placed can be used to illustrate how customer-specific constraints and ordinary case constraints interact XSEL generates this command when it knows either that the expected usage of some Unibus device that a customer needs will vary from the normally expected usage, or when the customer will be adding modules to the system and wants to leave space for those modules on the Unibus. The command identifies the module to be positioned by name; it specifies a slot number, a backplane position, a box. and a cabinet; it also indicates the amount of space that the module occupies To deal with this command. R1 has a number of rules that step in and nut of the regular configuration process at various points. Ordinarily when configuring Unibus modules, R1 first determines the optimal (normal usage) sequence for those modules and then attempts to place the modules in that sequence. The placement task involves selecting a module, a box (if there is more than one Unibus), a backplane (on the basis of the backplane requirements of the module selected), and additional modules until the backplane is filled. There are rules at each of these points that take control away from the normal configuration process if working memory contains a command to position a Unibus module in the box or backplane being filled Before the modules are placed in an optimal sequence, R1 marks as not to be included in the optimal sequence each module wnose position has been specified by XSEL. When selecting the box and module to configure next, a module specified to be placed in the next backplane position in a box will be selected in preference to the next module in the optimal sequence. When selecting a backplane for a module, R1 makes sure that the backplane selected is of the appropriate type and size. Finally, in filling up a backplane, R1 places any other modules that it has been told to put in that backplane before placing the remaining modules.

The number of rules that had to be added to realize the ad hoc constraint-handling capability was quite small relative to the total number of rules. R1 has 760 rules that enable it to do all but the

floor layout task; of these, 87 are rules which recognize commands generated by XSEL. These rules essentially overlay RVs general configuration rules; they know how to modify partial configurations (or other data structures) in a way that changes the state out from under R1 without affecting RVs ability to recognize what action is appropriate next. This extension of R1 is perhaps as interesting in what it shows about the inherent robustness of the Match method as it is as a capability in itself. A few (about 20) of RVs rules were changed in adding the new capability, but the changes were all in the direction of further specifying the conditions under which the old rules were applicable; that is, in a few cases, it was necessary to add a condition element to the old rules to give the new rules an opportunity to step in and take control, but essentially the old rules remained unchanged.

It should be noted that in the course of adding the capability, it became obvious that the idea of communicating by command could be exploited by R1 itself. In some cases, RVs processing provides it with information that constrains the configuration in some way before it is quite ready to use that information. The commands provide a convenient way of storing the information for subsequent use. For example, early in its processing R1 checks the baud rate of certain Unibus modules to insure that total baud rate does not exceed the maximum permitted. If the maximum is exceeded,*R1 marks the extra modules as not to be configured. However, if there is more than one Unibus on the system, more modules can be configured provided that they are distributed among the Unibuses. R1 discovers the excessive baud rate well belore it begins to configure Unibus modules; it can now at this point generate a command specifying on which Unibus each of the modules is to be placed, thereby simplifying considerably the subsequent box and module selection task.

3. A Foundational Ad Hoc Constraint-Handling Capability

The floor layout task differs from the other tasks involved in configuration in that it is impossible to do an acceptable job of floor layout without some customer specific information. Since R1 has had no access to such information, a floor layout capability was not included initially. During the Fall of 1980, the version of R1 that will be used in conjunction with XSEL was provided with a floor layout capability.[2] R1 is now capable of producing diagrams for multiple rooms showing the position and orientation of each component. Because the floor layout task presupposes some customer-specific information, our approach to developing an ad hoc constraint-handling capability for the floor layout task differed significantly from our approach to handling other sorts of ad hoc constraints. After describing RVs current knowledge of how to do floor layout, these differences will be discussed.

Much of the design and implementation of the new floor layout capability was done by Brigham Bell.

3.1. RVs Floor Layout Capability

Though R1 assumes that XSEL will provide it with room dimensions and with the locations of doorways and obstructions, it does not rely on XSEL for any other information. To perform the floor layout task, R1 first groups components on the basis of cabling considerations and then places these groups on the floor in ways that satisfy the ordinary-case configuration constraints it knows about Before trying to place a group of components, R1 creates a working memory element to represent each obstruction-free rectangular space in the room. Of the spaces that are large enough to contain the group, R1 selects that space which is furthest from the busiest door in the room. If none of the spaces are large enough, it backtracks. For example, suppose a set of rectangular spaces tor some group, A, is generated, and group A is placed in one of them. Next, a set of spaces is generated for group B, but none are large enough to contain it. Then group A, the last group of successfully placed components, would be removed from the room, and the working memory element which represented the rectangular space group A occupied would be deleted. Of the remaining spaces originally generated for group A, the next best space (i.e., the space furthest from the busiest door) would be selected, and group A placed in that space. A new set of spaces would then be generated for group B in the hope that one of these spaces will be large enough to contain it. Larger groups (those with more components) are positioned before smaller groups, and groups with many constraints on where they can be placed are positioned before less constrained groups.

The fact that R1 is generating a floor layout for a computer system rather than for some other set of objects does not yet affect its configuration strategy to a significant extent. R1 does assume that terminals and line-printers should be placed near the busy door, that all other components should be placed as far from busy doors as possible, that components should be placed in rows with certain aisle and service space restrictions, that all of the components in the same row should face in the same direction, and that rows of components should face each other. When there is unused space in a room, rows facing each other are pushed apart. These are all of the domain-specific knowledge that RI currently has for performing the floor layout task.

3.2. Customer Preferences and the Use of Ordinary-Case Constraints

The commands that XSEL can generate to affect the way components are laid out on the floor are comparable in power and scope to the other commands it can generate. In the case of floor layout, however, XSEL acts simply as an intermediary; it Dasses to R1 whatever floor layout ideas a customer proposes. In

interacting with XSCL, a customer can specily exactly how he wants his system to be laid out; that is. he can specify the location and orientation of each component. It he wishes, a customer can constrain the la/out without specifying exactly where each component is to be placed. He can indicate an area in a room where a group of components is to be placed. He can indicate that certain groups of components are to be placed in the same room. Or he can indicate that a particular area in a room is to be left vacant.

Two things have become evident during the brief period in which the floor layout capability has been used. Our first discovery was that in most cases customers want to specify, at least in general terms, where components are to be placed. The second discovery (which came as no surprise) was that RVs knowledge of ordinarily applicable floor layout constraints is quite incomplete. Taken together, these two pieces of information suggest that the relationship between customer-specific and ordinary case constraints, as they apply to floor layout, will have a quite different character than in the rest of the configuration task. In floor layout, RVs ordinary case knowledge is in a sense superimposed on the knowledge of how to deal with customer specific constraints.

Though R1's floor layout capability does not presuppose customer generated constraints, it uses such constraints, when they are available, to limit the space it must search. RVs current knowledge of the ordinary case constiaints on floor layout is so incomplete that when it is given no help from the customer, the floor layout it produces may have serious inadequacies. As floor layout knowledge continues to be extracted from the experts and given to R1, it is likely that R1 will be able to produce consistently adequate layouts. But it is quite clear that it will be unable, except by chance, to produce on its Qwn a layout that the customer finds completely satisfactory. Consequently RVs role with respect to floor layout is likely to almost always be one of refiner. R1 will use its knowledge to complete a design proposed by the customer. Thus the task of providinq R1 with knowledge of ordinarily applicable floor layout constraints will be comparable to the task of providing it with its general ad hoc constraint handling capability. The rules which comprise its knowledge of floor layout will watch for situations in which a customer has not fully specified something, will step in and provide a constraint, and then step back out to let the now more fully constrained layout be produced.

4. Concluding Remarks

With the addition of relatively few rules, a system that was unable to cope with ad hoc constraints has been transformed into a system that is quite flexible. Though RVs newfound flexibility does not extend beyond the boundaries of the configuration task, the change in its competence is marked. It is interesting that R1 has become more expert without acquiring more knowledge of the configuration domain. R1 has the same knowledge it always had: knowledge which does not give it a broad understanding of the various trade-offs within the space of acceptable configurations. But in addition, it now has knowledge that enables it to adapt to externally imposed restrictions. Thus, even though it does not know enough to determine the configuration-related implications of the intended use of a system, it does know how to follow advice from someone whose knowledge of configuration design complements its own.

Acknowledgments

Several colleagues at CMU have provided valuable suggestions and criticisms, in particular, Brigham Bell, Barbara Chessler, Tom Cooper, Charles Forgy, Allen Newell and Guy Steele. Sam Fuller, Arnold Kraft, David Livingstone, Dennis O'Connor, and many others at Digital have been a constant source of encouragment.

References

[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*, pages 933-939. MIT, 1977.

[Forgy 80] Forgy, C. L. *The OPS5 user's manual.* Technical Report, Carnegie Mellon University, Department of Computer Science, 1980.

[McDermott 80a] McDermott, J. *R1: a rule-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*, pages 269-271. Stanford University, 1980

[McDermott 81] 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*, pages 361-414. John Wiley and Sons, 1969