CMU-CS-80-119

R1: A RULE-BASED CONFIGURER OF COMPUTER SYSTEMS

John McDermott

April, 1980

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

90 07 16 460

CMU-CS-80-119

R1: A RULE-BASED CONFIGURER

OF COMPUTER SYSTEMS

John McDermott

April, 1980

Abstract. R1 is a program that configures VAX-11/780 computer systems. Given a customer's order, it determines what, if any, modifications have to be made to the order for reasons of system functionality and produces a number of diagrams showing how the various components on the order are to be associated. The program is currently being used on a regular basis by Digital Equipment Corporation's manufacturing organization. R1 is implemented as a production system. It has sufficient knowledge of the configuration domain and of the peculiarities of the various configuration constraints that at each step in the configuration process, it simply recognizes what to do. Consequently, little search is required in order for it to configure a computer system. (KR)

The development of R1 was supported by Digital Equipment Corporation. The research that led to the development of OPS4, the language 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 under Contract F33615-78-C-1151. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Digital Equipment Corporation, the Defense Advanced Research Projects Agency, or the U.S. Government. VAX, PDP-11, UNIBUS, and MASSBUS are trademarks of Digital Equipment Corporation.

INTRODUCTION

R1[1] is a rule-based system that has much in common with other domain-specific systems that have been developed over the past several years [Amarel 77, Waterman 78]. It differs from these systems primarily in its use of Match rather than Generate-and-test as its central problem solving method; rather than exploring several hypotheses until an acceptable one is found, it exploits its knowledge of its task domain to generate a single acceptable solution. R1's particular area of expertise is the configuring of Digital Equipment Corporation's VAX-11/780 systems. Its input is a customer's order and its output is a set of diagrams displaying the spatial relationships among the components on the order; these diagrams are used by the technician who physically assembles the system.[2] Two inter-dependent activities must be performed in configuring a VAX system.

- The customer's order must be determined to be complete; if it is not, whatever components are missing must be added to the order.

- The spatial relationships among all of the components (including those that are added) must be determined.

The criterion of success for whether a configuration is complete does not reside in any simple test, but involves instead particular knowledge about all the individual components and their relationships. The criterion of successful spatial arrangement is more compact (reflecting the uniform character of geometric structure), but it too involves particular knowledge on a component by component basis. Thus, the task accomplishment is defined by a large set of constraints embodying a large amount of knowledge.

Although a significant portion of this paper is devoted to a description of precisely how R1 goes about doing the configuration task, I have tried to avoid letting the details of R1's inner workings overshadow the domain independent lessons that have emerged from this research. There are two important lessons:

- Recognition knowledge can be used to drive an expert system's behavior, provided that it is possible to determine locally (ie. at each step) whether taking some particular action is consistent with acceptable performance on the task.

- When an expert system is implemented as a production system, the job of refining and extending the system's knowledge is quite easy.

The paper is divided into three sections. The first section describes the VAX-11/780 configuration task and characterizes its difficulty. The second section describes R1 and discusses its evolution

[1] Four years ago I couldn't even say "knowledge engineer", now I ...

[2] R1's output for a sample order is shown in Appendix 2.

from a system with only the most limited capabilities to what might fairly be called, a true expert. The third section describes R1's current level of expertise and isolates the design decisions that made the building of R1 straightforward.

1. THE TASK

The VAX-11/780 is the first implementation of Digital Equipment Corporation's VAX-11 architecture. It is similar in many respects to the PDP-11, though its virtual address space is 2^{32} rather than 2^{16}. The VAX-11/780 uses a high speed synchronous bus, called the sbi (synchronous backplane interconnect), as its primary interconnect. The central processor, one or two memory control units, one to four massbus interfaces, and one to four unibus interfaces can be connected to the sbi. The massbuses and particularly the unibuses can support a wide variety of peripheral devices. Because the number of system variations is so large, the VAX configuration task is non-trivial.

1.1. THE SIZE OF THE TASK

A configurer must have two sorts of knowledge. First, he must have information about each of the components that a customer might order. For each component, the configurer must know the properties that are relevant to system configuration -- eg, its voltage, its frequency, how many devices it can support (if it is a controller), how many ports it has; I will call this knowledge *component information*. Second, he must have rules that enable him to associate components to form partial configurations and to associate partial configurations to form a functionally acceptable system configuration. These rules must indicate what components can (or must) be associated and what constraints must be satisfied in order for these associations to be acceptable; I will call this knowledge *constraint knowledge*.

The difficulty of the VAX configuration task is a function of the amount of component information and the amount of constraint knowledge required to perform the task. It is fairly easy to estimate the amount of component information that is needed. On the average, a configurer must know eight properties of a component in order to be able to configure it appropriately. Currently about 420 components are supported for the VAX.[3] Thus there are over 3300 pieces of component information that a VAX configurer must have access to.

Before R1 was developed, it would have been difficult to estimate accurately the amount of constraint knowledge required for the configuration task. Much of the required knowledge was not

[3] Of the 420 components, about 180 are actually bundles composed of various subsets of the remaining 240 components.

written down anywhere and thus the only source of estimates would have been individual human experts. But the experts find the task of quantifying their constraint knowledge foreign. As I extracted this knowledge from them, it became clear that their knowledge takes two forms: (1) The experts have a sparse but highly reliable picture of their task domain. When asked to describe the configuration task, they do so in terms of the subtasks involved and the various temporal relationships among these subtasks. (2) They also have a considerable amount of very detailed knowledge that indicates the features that particular partial configurations and unconfigured components must have in order for the partial configurations to be extended in particular ways. Both sorts of knowledge are easily expressable as rules. I extracted 480 rules. Of these, 96 define situations in which some subtask should be initiated. The other 384 rules define situations in which some partial configuration should be extended in some way.

1.2. THE CONSTRAINTS

This subsection provides two examples of specific subtasks that can arise within the configuration task and indicates for each (1) the constraint knowledge involved, (2) the informational demands imposed by that constraint knowledge, and (3) the extent to which the subtask presupposes other subtasks. The first subtask is to place unibus modules into backplanes; the second is to assign massbus devices to massbuses.

Example: Placing unibus modules in backplanes. Whenever more than one unibus option is ordered for a VAX, it is necessary to place the modules on the unibus in an acceptable sequence. It is straightforward to determine the optimal sequence for the modules; the modules are sorted on the basis of their interrupt priority and within that on the basis of their transfer rate. Before a module can be placed on the unibus, it is necessary to select a backplane. Several constraints come into play. Backplanes come in two sizes (4-slot and 9-slot) and can have any of several pinning types. The backplane selected must be of the pinning type required by the unibus module. To determine the size of the backplane to be selected, it is necessary, first, to determine whether the size is constrained by the box that the backplane will be placed in. A box can accommodate five 4-slot backplanes. In most cases a 9-slot backpane may be used in place of two 4-slot backplanes; the exception is that a 9-slot backplane may not occupy the space reserved for the second and third 4-slot backplanes.[4] Assuming that either a 4-slot or a 9-slot backplane would be acceptable, the next constraint to come into play is that a 9-slot backplane should not be selected unless the next N modules in the optimal sequence all require a backplane of the same type and will not all fit in a 4-slot backplane. Once a backplane is

---

[4] The box that contains unibus modules has two +5 volt regulators. One of these regulators supplies power to the first two 4-slot backplanes (or to the first 9-slot backplane); the second supplies power to the other backplanes. All of the modules in a backplane must draw power from the same regulator.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming inter-face) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE