YES/MVS and the automation of operations for large computer complexes

by K. R. Milliken
A. V. Cruise
R. L. Ennis
A. J. Finkel
J. L. Hellerstein
D. J. Loeb
D. A. Klein
M. J. Masullo
H. M. Van Woerkom
N. B. Waite

The Yorktown Expert System/MVS Manager (known as YES/MVS) is an experimental expert system that assists with the operation of a large computer complex. The first version of YES/MVS (called YES/MVS I) was used regularly in the computing center of IBM's Thomas J. Watson Research Center for most of a year. Based on the experience gained in developing and using YES/MVS I, a second version (YES/MVS II) is being developed for further experimentation. This paper discusses characteristics of the domain of large computing system operation that have been illuminated by the YES/MVS I experience, and it describes the modifications in the design of YES/MVS II that are an outgrowth of the YES/MVS I experience.

An *expert system* is a computer application that uses explicitly represented knowledge and computational inference techniques to achieve a level of performance comparable to that of a human expert in some application area or *domain*. Expert systems have been developed for a variety of domains, such as diagnosing blood diseases,[1] oil drilling,[2] geological exploration,[3] and determining chemical structures.[4]

The domain of the expert systems that are the subject of this paper is the operation of large computing systems, or more specifically, the operation of one or a cluster of IBM mainframe computers, each controlled by the IBM Virtual Storage 2 Multiple Virtual Storage Operating System (OS/VS2 MVS, or simply MVS).[5] We refer to the computing system or cluster being operated as the *target* system. The target systems considered to date have all provided computing services to interactive users via the Time Sharing Option (TSO) of MVS and to a group of submitted jobs under the control of JES3 (the Job Entry Subsystem 3 of MVS).[6,7]

A cluster of systems typically has several operator consoles, for example, consoles for operation from a central point, in the tape library, near tape drives or printers, or in the offices of operations supervisors and systems programmers. Nevertheless, MVS and JES3 cooperate in a cluster so that most of the complex aspects of system operation can be performed for the entire cluster from just one JES3 console.

The Yorktown Expert System/MVS Manager (YES/MVS) is an experimental expert system for the do-

main of MVS/JES3 operation. It addresses the problem of the operation of a cluster of MVS/JES3 systems through one JES3 console and provides advice on manual intervention needed to solve operational problems. A first version, called YES/MVS I, was regularly used in the computing center of IBM's Thomas J. Watson Research Center for most of a year. YES/MVS I assisted with the operation of a single MVS/JES3 system running on an IBM Model 3081K processor. From the experience gained with YES/MVS I, a decision was made to develop a second version, YES/MVS II, that would significantly improve on the breadth and depth of function provided by YES/MVS I. More technical detail on YES/MVS I can be obtained from three papers by Ennis et al.[8-10]

It is the intent of this paper to review the function, organization, development process, and experience gained in the use of YES/MVS I as a basis for discussing the lessons that have been learned from that effort. We shall concentrate on lessons that appear to be important for ongoing attempts to automate large system operations.

The first section of this paper introduces the operation of an MVS/JES3 system, the function and organization of YES/MVS I, and the expert-systems language OPS5 (used in the development of YES/MVS I). The second section provides insights into pertinent YES/MVS I experiences, such as the nature of an operator's knowledge, the value of using rules for developing expert systems, and the need for maintaining a status model of the target system.

The third section discusses the ongoing work on YES/MVS II. Much of the work is guided by the following observation: An expert operator at one center would probably not immediately be an expert at another but would require a period of training to become knowledgeable about the characteristics of operations that are unique to the second center. This situation is due to the necessary and inevitable variation among centers, variation due to differences in hardware configuration, installed software, workload, and operational policy. An expert system that assists with the automation of operations encounters the same variation; customization for individual sites, then, is a major challenge of automated operations. This section includes a discussion of the design features of YES/MVS II intended to facilitate customization—specifically, better representation of operational knowledge, automatic maintenance of a status model of the target system, generally applicable knowledge of operational problems, techniques for testing an operational knowledge base, and approaches to knowledge acquisition.

Motivation for an expert-systems approach to automated operations

Operation of an operating system. During the last twenty years, facilities have been developed to control most of the resource allocation that is done on a routine basis in large computing systems. There exist subsystems that prioritize, schedule, assign resources to work, etc., and operators increasingly are not required to perform these functions. While MVS and other operating systems have a growing number of built-in problem-handling and recovery mechanisms, the court of last resort continues to be the operator.

Total computing power is growing rapidly.

Basically, operators have remained necessary (for tasks other than the merely manual ones) because of a wide variety of complex system problems that can occur and that, in turn, depend for resolution on data or knowledge not normally available to the operating system: e.g., what work is to take precedence in certain problem situations, which system resources are critical to the services being provided by the system, and whether any of the operators currently in the center are trained to perform some manual task. Solving the more serious problems requires that the operator (a) monitor the system and maintain a mental model of its state, (b) recognize symptoms of problems, (c) obtain additional data on the cause of symptoms, (d) diagnose genuine problems, and (e) take appropriate steps to solve problems. To be done well, the process requires considerable knowledge and training.

In addition, the operator's job is becoming more challenging: First, total computing power is growing rapidly at many computing centers. This growth brings a corresponding increase in console message

rates (number of messages per second an operator must handle), which for some large clusters are already uncomfortably high. Second, since the growth rate in total computing power at many centers is exceeding the increase in uniprocessor speeds, large computing centers have installed clusters of processors; this distribution of the resources has made the systems more complex in all respects. Third, as the importance of system availability has grown, so has the need for fast and accurate operator actions in response to system problems that threaten system availability. In short, the operation of large systems is complicated by several features: high message rates, multiplicity of consoles, the complexity inherent in providing problem diagnosis and response, the requirement for fast and accurate response to problems, the variation in hardware, software, workloads, and policy among computing centers, and the evolving nature of computer systems.

Because of this complexity, training a skillful operator requires significant time, and expert operators are certainly not plentiful. An obvious response is to move toward automated operations. If successful, automated operations should provide increases in labor productivity, system availability, and management control.

To date, several projects have made progress in automating operations, e.g., CCOP (the Centralized Computer Operation Project).[11] Goals have included simply filtering out unwanted messages, centralization of operations, and active response to particular messages generated by the target system. Generally, projects not based on expert systems have used many loosely organized modules of procedural code to respond to individual messages. The weaknesses with procedural approaches, and, in turn, the motivation for the use of expert-systems techniques, are due to (a) the basic complexity of the problem domain and (b) the requirement for flexible software to meet the needs for reasonably easy customization at installation time and for ongoing maintenance of any facility that assists with the knowledge-intensive parts of an operator's responsibilities.

The function and organization of YES/MVS I. YES/MVS I is a research prototype developed to investigate the feasibility of automating the complex portions of the job currently performed by human operators for MVS computer systems. In particular, the goal of YES/MVS I was to obtain high-quality results in selected, challenging areas of operator activity but not to construct a comprehensive system that would automate all MVS operations. The areas selected were ones that were known to be challenging and were of practical interest in the computing center at the Watson Research Center.

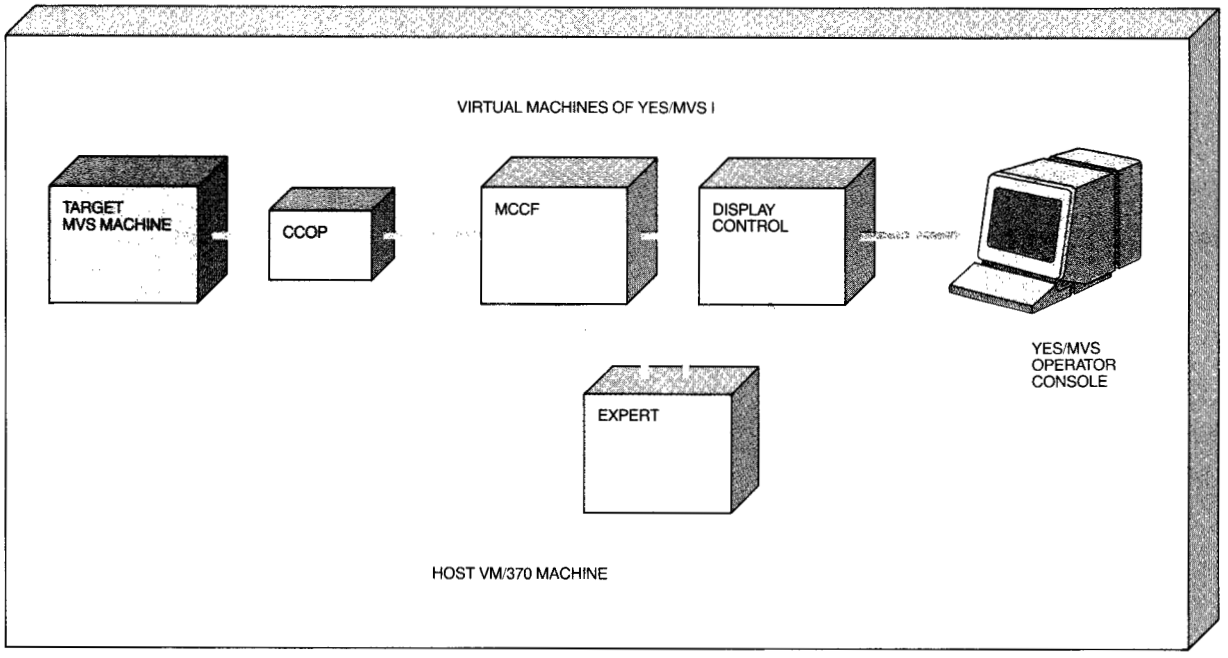
To MVS, YES/MVS appears to be a very fast human operator in that YES/MVS submits commands to and receives messages from MVS. The organization of YES/MVS is shown in Figure 1. YES/MVS runs in a separate computer from the target system. This separation has primarily been made to avoid having YES/MVS dependent on the target for computing resources, especially when the target is not fully operational. The interface between YES/MVS and MVS is a JES3 console which appears to MVS as a standard operator's console. However, the console can be "read" and "typed on" by YES/MVS. YES/MVS I runs in three concurrent virtual machines under the Virtual Machine/System Product (VM/SP) operating system.[12]

1. *The Expert virtual machine.* The heart of YES/MVS is the Expert virtual machine. The Expert runs a program based on (developed from) the knowledge of expert human operators. The program is written in an extended version of the expert-systems language OPS5 which, in turn, runs under LISP/VM.[13] (The next section contains more details on OPS5.) The Expert virtual machine contains all of the crucial data needed by YES/MVS, for example, all data required to model the state of the target system. The Expert communicates with the target through the virtual machine MCCF and, with the operator, through Display Control.

   A *subdomain* is a related, largely separable area of operator activity such as monitoring channel-to-channel links, batch job scheduling, or managing JES queue space. All the rules for different subdomains of the operator's actions coexist in one knowledge base. Running expertise from multiple subdomains allows sharing of status information, and gives the expert system control over the scheduling of actions in different subdomains, rather than leaving such scheduling to the underlying VM operating system.

2. *The MVS Communications Control Facility virtual machine.* A second virtual machine runs the MVS Communications Control Facility (MCCF), which is written in the REXX language[14] and in assembly language. MCCF controls the receipt of messages from MVS and the formatting of commands specified by the expert system. MCCF re-

Figure 1 Organization of YES/MVS I



ceives messages and submits commands through the CCOP facility.[11] Thus, the expert system is effectively insulated from the format of MVS and JES messages/commands, and the equivalent information is passed to/received from the Expert.

MCCF provides a table-driven match and translate capability. The desired messages are described in tables, the fields containing variable parameters are specified, and a description of the desired output data structure is included. When a desired message arrives, it is identified and translated, and the corresponding data structure is sent on to the Expert virtual machine. Arriving messages that are of no interest are discarded. MCCF also builds and submits commands to MVS upon receipt of a command name and associated parameters from the Expert.

3. *The Display Control virtual machine.* A third virtual machine, Display Control, also controlled by software written in OPS5, provides the communications interface between the human operator and YES/MVS. The display provides a hierarchically structured interface to the operator. At the highest level, operators are alerted to problems and notified of their severity. When a specific problem is encountered, the operator may select a lower level to obtain more detail on the problem. Explanations are available on the nature and severity of a problem, as are precise descriptions of suggested actions. The display also presents current values of various system parameters, e.g., the number of batch initiators. To change a parameter value, the operator merely overwrites it; YES/MVS senses the change and automatically submits the command necessary to effect the change.

We found the use of separate virtual machines to be extremely useful since the functions of the three machines are naturally asynchronous. Indeed, YES/MVS II employs a similar design.

The principal subdomains automated in the Expert component of YES/MVS I are as follows:

- Scheduling large batch jobs off prime shift: This scheduling must be done in a manner that appropriately considers user satisfaction, system throughput, and installation priorities.
- Managing JES queue space: Before, during, and after execution, all jobs processed under MVS are staged from a central spool file called the JES queue

space. Operators must carefully monitor JES queue space since JES cannot recover if this space is exhausted.

- Problems in channel-to-channel links: Often computers are interconnected via channel-to-channel links. Failure to maintain these links in an active status not only delays data traffic but also can contribute to the exhaustion of JES queue space.
- Hardware errors: When MVS is unable to recover from a hardware error, the operator is notified. A timely response (e.g., reconfiguration) is often required to avoid a total system failure.
- SMF management: The System Measurement Facility (SMF) is an MVS subsystem that provides access to information on resource utilizations. There are several routine actions that must be taken to manage SMF data.
- Quiesce and Initial Program Load (IPL): Before a planned shutdown (e.g., to install new hardware), the target system must be *quiesced.* The quiesce operation typically takes 30 minutes and involves many operator actions. Since YES/MVS I does not have access to the system console (only a JES3 console), it cannot trigger an actual IPL. However, YES/MVS I does give advice on IPLs, and once the JES console becomes operational, YES/MVS I is able to take over.

YES/MVS I operates in two modes: *advisory* and *active.* In advisory mode, YES/MVS obtains from the target system all information necessary to perform an activity. However, instead of actually submitting a command that would change the state of the target (e.g., vary a device off line), YES/MVS displays the command it recommends submitting along with an explanation. If the operator agrees with the recommendation, YES/MVS submits the command. The advisory mode is extremely useful for debugging. Once operators trust YES/MVS, active mode is normally used. In active mode, YES/MVS automatically takes action without consulting the operator, although explanatory information is still made available to the operator.

An introduction to production systems and OPS5. In YES/MVS, the knowledge to control operations is written in a LISP-based version of the production system language, OPS5.[15,16] This section briefly discusses production systems in general and OPS5 in particular.

Productions or *rules* are if-then statements. The if-portion or *left-hand side* (LHS) describes a situation; the then-portion or *right-hand side* (RHS) specifies actions to take when the LHS situation arises. A *data-driven* (or *forward chaining*) *production system* operates by finding rules whose if-portions are satisfied and then executing the then-portion of one such rule. When an RHS is executed, the rule is said to have *fired.* The sequence of rule firings is controlled by a special run-time package called the *inference engine* which is responsible for determining which rules have their LHS satisfied as well as selecting a rule to fire. Thus, by using rules, the programmer is relieved of many flow-control issues, which can be extremely troublesome in conventional programming languages. English-language examples of typical rules appear in the later subsection on the value of the rule-based approach and in the one on generally written knowledge.

OPS5 is a data-driven production language. An expert system written in OPS5 has three components: *working memory,* in which temporary data are stored, the *rule base* or *rule memory* (consisting of all the productions), and the inference engine. Working memory is partitioned into classes of *working-memory elements*; all elements in a given class have the same data format. A working-memory element (WME) consists of several fields or attributes. The data format for a class is defined by the *literalize* construct. For example,

```
(literalize  printer-status      ; Name of working-memory class
             status-current?     ; Is the status current?
             status)             ; Status of printer
```

The second component of an OPS5 expert system, the rule base, contains the production rules. In OPS5, a situation or LHS is expressed as a query on working memory. A query may reference several WMEs, each of which is called a *condition element.* The following is a sample OPS5 rule.

```
(p jm:printer-status-update                ; Name of rule
   { (message) (printer-status-reply       ; Reply from MVS to query
     ↑ address (printer-address)           ;   about printer status
     ↑ status (new-status))}
   { (printer) (printer-status             ; Status WME for printer
     ↑ address (printer-address))}         ;   with same address as reply
→
   (remove (message))                      ; Remove the reply
   (modify (printer)                       ; Update the printer status:
     ↑ status (new-status)                 ;   -set new status
     ↑ status-current? yes))               ;   -status now current
```

This rule detects when MVS has sent a reply to a command that inquires about printer status. On the basis of the reply, the rule updates the WME that models printer status. The LHS consists of two con-

dition elements: one of class "printer-status-reply," and one of class "printer-status." The condition elements appear above the right-pointing arrow. These conditions must be satisfied; that is, particular instances of the two WMEs must be in working memory for the rule to be considered eligible by the inference engine. Furthermore, the "printer-status-reply" WME and the "printer-status" WME have mutual conditions that must be satisfied in that the value of their respective "address" attributes must be the same.

If the inference engine selects a rule, the actions on the RHS of the rule are executed. Typically, these actions involve changing working memory by creating new WMEs with the MAKE function, modifying existing WMEs with the MODIFY function, or removing an existing WME with the REMOVE function; LISP functions may also be used. In the case of the rule "jm:printer-status-update," there is both a modify and a remove. The REMOVE eliminates the "printer-status-reply" WME used to satisfy the first condition element; the MODIFY refers to the "printer-status" WME used to satisfy the second condition element.

The OPS5 inference engine has three phases: *recognize*, *conflict resolution*, and *act*. In the recognize phase, the LHSs of all rules in the knowledge base are compared with the contents of working memory to determine which rules have their condition elements satisfied. A rule may have its LHS satisfied by more than one set of working-memory elements. Each combination of a rule and a set of matching working-memory elements is called an *instantiation*. The set of all instantiations is called the *conflict set*. If the conflict set contains more than one instantiation, the inference engine selects one; this selection process is referred to as *conflict resolution*. Since comparing all condition elements (in the recognize phase) against working memory can be computationally expensive, OPS5 uses a special algorithm called the RETE matching process[17] which improves performance by (among other things) incrementally maintaining a list of all instantiations in the conflict set as each rule is fired.

To develop YES/MVS I, some extensions to OPS5 were necessary. We give a brief overview here. For more details, consult Ennis et al.[10] First, timed reminders are needed to perform operations such as periodic queries of the target system and to check for nonresponses. A special kind of MAKE, called TIMED_MAKE, was developed. TIMED_MAKE creates a WME at a future time. Thus, checking for a nonresponse requires (1) doing a TIMED_MAKE for a predetermined class (say X) at the desired time and (2) writing a rule whose LHS is satisfied when a WME of class X is created and whose RHS determines whether a response has arrived.

A second facility required for YES/MVS is communication among virtual machines, since the Expert, Display Control, and MCCF components all reside in different virtual machines. This requirement is met by providing another type of MAKE called REMOTE_MAKE, which creates a WME on another virtual machine. Thus, message passing is quite simple: (1) The sender does a REMOTE_MAKE of a WME for a predetermined class (say Y) and provides the destination address. (2) The recipient typically has a rule whose LHS is satisfied when a WME of class Y is created.

Lessons learned from YES/MVS I

In comparison with previous efforts to automate operations, the principal new factor in YES/MVS is the use of expert-systems software technology. Using rule-based software, the YES/MVS effort has concentrated on automating the more complex aspects of the operator's job: resource allocation and scheduling, and especially problem detection, diagnosis, and containment or recovery. The fundamental lesson from the YES/MVS I effort is that this approach worked. The YES/MVS I knowledge base was developed by a reasonably small group of people over a period of less than two years. With that knowledge base, the facility was successful at automating a variety of complex tasks performed by operators at the computing center of IBM's Thomas J. Watson Research Center.

During the development and use of YES/MVS I, a number of other lessons have been learned. (See also Ennis et al.[10] and Schor.[18]) The rest of this section outlines facets of the operator's job and of operational knowledge that must be considered during the automation of operation. Also, we review some key features of YES/MVS I in light of the characteristics of large-systems operations.

An operator's knowledge. Typically, operators sit at a console; they submit queries; and they watch for responses to their queries as well as for system-generated messages. When an action is required, they submit the appropriate command(s). Most system-volunteered information is in single messages. Responses to queries are frequently in the form of multiline messages, in which case there is a sequence of lines on the operator's screen taken up by a

response to one query. Sometimes the lines are similarly structured but have varying fields. This is the normal case when, for example, one inquires about all the jobs in the system satisfying certain criteria. Each line contains information about one particular job that satisfies the conditions of the query. For other multiline messages, there is more information than can fit in one line, or the information is organized so that it is best conveyed on multiple lines.

Most of an operator's actions can be divided into three categories.

When looking at the screen of an operator's console, it is fairly easy to keep track of the various pieces. Visually, the multiline messages are similar enough so that one naturally groups them together. Also, the order in which active commands are submitted is the order in which responses are received, and the order in which queries are submitted is the order in which responses are received. Thus, if an operator waits for one response before submitting another command that could give a similar response, there is no difficulty in deciding what response goes with what query. These things are easy for humans, but they are rather difficult for computer programs because each line on the console cannot be taken as an individual entity. It must be taken and understood in the context of the nearby messages and commands. The program must contain logic to provide the visual decomposition of the screen and the temporal relationships that are fairly naturally provided by an experienced human.

Most of an operator's actions can be divided into three categories: monitoring, resource allocation, and problem handling. The monitoring portion of the operator's job consists predominantly of gathering information (e.g., submitting queries to get the current level of JES queue space), identifying and following trends, and detecting problems either by comparing values against thresholds or by more complex means when historical information must be considered. Note that effectively monitoring the system requires having a conceptual model of system services and resources, such as the rate at which status of a resource can change, and how critical a resource is to the system and the services being provided. In addition, an operator must know what queries to submit to obtain system status information, what other status information is available, and how one identifies symptoms and detects problems from the available status information.

The dominant activities classified as resource allocation or management are work scheduling and manually providing resources for input and output (I/O) devices (tape mounting, supplying paper for printers, etc.), though timed and workload-triggered preventive maintenance and tuning are also common. The amount of responsibility that operators have over scheduling of work varies widely with the policy of the computing center. Work scheduling normally involves maintaining prioritized queues of jobs to be done. Again, the operator must have an understanding of the workload, priorities, deadlines, required resources, available resources, queries and available information, active commands, and unusual circumstances. In complex situations, the operator is required to prioritize multiple, conflicting considerations (resource utilization, response time, age, total wait time, priority of users and classes of work); he must plan ahead to match expected demands with resources; he must monitor and dynamically adjust to changes in workload, available resources, and unusual circumstances.

For some types of problems, the correct or best response is obvious once diagnosis is successfully made, and most corrective actions taken at a console involve only one or a short sequence of commands. Physical intervention may be required to turn power on to devices, load microcode into devices from a floppy disk, remove printer jams, or the like. Frequently, increased monitoring is indicated after the corrective commands to ensure that the problem is corrected and does not recur. For example, if a noncritical subsystem fails, the typical action after diagnosing the situation is to restart the subsystem and then monitor it carefully to determine whether the problem will repeat.

For other problems, the response may involve

1. A sequence of actions interspersed with decisions or monitoring, e.g., when restarting (by IPL) an entire system or cluster.
2. Selecting one or several actions from a collection of responses, each of which will help.

3. A trial-and-error approach because of insufficient knowledge about the situation, with possible corrective actions selected from a "bag of tricks."

However, under most circumstances, there is pressure on operators to diagnose and respond to a problem quickly—before things get worse. This pressure combined with the complexity and the constantly changing nature of system status makes problem handling an error-prone process. Typing errors alone can be significant. Operator errors can and occasionally do make problems worse.

Illustrative example: JES queue space management. In this subsection we shall outline the strategy encoded in YES/MVS for handling a particular type of operational problem—JES queue space depletion. Later in the paper, we will use specific characteristics of this example to provide concrete illustration of certain more general and abstract points that are important in the paper.

JES queue space is a common resource (disk storage) in MVS/JES3 systems for the staging of computer jobs before, during, and after execution. Jobs are normally deleted from the queue space once output has been completed to a printer, a communication link, or other output medium. JES queue space is also used by JES itself as a scratch area for executing its functions. In addition, JES maintains batch job output for on-line viewing (via TSO) in the JES queue.

Operators are concerned with monitoring the remaining available queue space because its exhaustion requires restarting the system and inconveniencing all system users. The operator may take several protective and corrective actions when queue space begins to diminish, and these may be described in terms of three general goals:

- Protect remaining queue space: The operator must protect the space that remains when it has become dangerously low (e.g., less than five percent remains available).
- Free queue space: The operator can manipulate various devices, operating-system parameters, or jobs in the work stream to free queue space.
- Diagnose and eliminate the cause(s) of queue space depletion.

The JES queue space problem-handling component of YES/MVS is responsible for monitoring JES queue space, detecting when a problem occurs, diagnosing its cause, and assisting the operator in solving queue space problems. Monitoring involves querying the target system for the remaining queue space at regular intervals, e.g., every five minutes. YES/MVS detects a JES queue space problem if the remaining space falls below a threshold. Several thresholds are used; a lower threshold indicates a more severe problem.

The JES queue space expert retains in working memory the response to its most recent query of JES queue space. If queue space begins to drop, the rate of periodic monitoring is increased. If the remaining space continues to go down, additional monitoring is initiated to maintain current status information on all the entities (e.g., printers) affecting the remaining level of JES queue space. When the problem is resolved and queue space returns to normal levels, the additional monitoring is terminated.

There are four general causes of JES queue space problems:

1. Lack of capacity (e.g., insufficient queue space, printer capacity, or communications network capacity to handle—in a long-term, steady-state sense—the amount of data being placed in the queue).
2. Failures in devices or subsystems that temporarily reduce the available capacity.
3. Suboptimal queue space utilization because of resource allocation policy (e.g., work of a particular print class, requiring a change of forms, might be saved until a fixed time each day to maximize productivity of operators).
4. An extraordinarily large amount of work or data in the queue because of unusual circumstances (e.g., a failing subsystem or job is in a loop and dumping data onto the queue, or a number of large dumps have been mistakenly left on the queue).

In a real problem situation, an implicit assumption is made that the steady-state capacity is not the problem. (However, repeated problems without reasonable solutions should be recognized as a symptom of insufficient capacity.) Problems rooted in combinations of the other three causes listed are still rich in variety, and problem diagnosis and recovery are frequently not straightforward.

If YES/MVS knows of actions involving better utilization of existing resources that would help with a queue space problem, those actions are attempted. Under the right circumstances, jobs will be rerouted

from overutilized to underutilized printers; line limits will be changed on printers, enabling large jobs to print that would normally have been held until another time of day; a change of printer forms will be suggested to operators; the printing of material with special security requirements will be suggested; or communication links will be restarted to allow material on the queue to be sent to its destination. Sometimes nonoperative devices and subsystems can be restarted, but frequently outside help must be sought for component repair.

It is always valuable to seek out and, where possible, to eliminate ongoing causes of queue space depletion. For example, runaway subsystems or jobs need to be found, as prompt corrective action is important. JES has its own mechanisms for catching and terminating runaway jobs, but runaway subsystems can happen, and such situations must be carefully diagnosed. If a subsystem is writing to a data file that has not been closed, then JES will report the space lost when the total space remaining is queried, but JES may not report the data file when queried about the contents of the queue.

Sometimes action is required which may not actually solve the queue space problem but is needed just to preserve the remaining queue space until the underlying cause is found. If the problem becomes moderately severe, YES/MVS assists the operator in moving queue entries to tape. YES/MVS selects the entries to move, and by using a special job (commonly called the DJ job), dumps entries to tape. Once the problem has been solved, queue entries can be restored. If the problem becomes critical, more drastic action may be needed to preserve queue space, such as varying the processor off line to JES until queue space returns to an acceptable level.

The rules in YES/MVS I that are dedicated to the management of JES queue space can be classified by function, as below. A similar classification holds for most subdomains of YES/MVS I.

- *System Initialization and Control:* These rules create and initialize the pertinent portion of the target system status model and otherwise initialize JES queue space management.
- *Periodic Query Submission and Timeout Handling:* This group controls the periodic querying of target system resources.
- *Information Collection and Data Reduction:* These rules collect target system messages and update the target system status model accordingly.
- *Miscellaneous Cleanup:* This rule group deletes target system responses and expert-systems-generated goals from working memory.
- *Knowledge-Based Action:* All rules in the above-described groups exist to support the Knowledge-Based Action Rules, which encode policy and expertise for managing JES queue space.

The control of problem diagnosis and resolution resides in the knowledge-based action group of JES queue space management rules. The best description of the approach to problem resolution is that it is "opportunistic." By this we mean that (as described above) there are a variety of actions that may help resolve a situation if their set of special prerequisites are met. Rule-based techniques support the writing of a number of separate rules, each with its LHS describing a list of special prerequisites for the action initiated by its RHS to be of value in resolving a situation. Hence, we can say that each rule acts like a *demon*, that is, a program component so-called because it "awakens" when conditions (problem severity, target system status, status of the goals and internal computations of YES/MVS) indicate that its action would be of value. Certain actions or changes in status enable other actions, but there is no global planning of a unified approach to a particular problem.

Among the knowledge-based action rules for managing JES queue space, a majority reflect local policy to some degree and might be unacceptable at another computing center unless modified. Essentially all the rules in YES/MVS that, during a queue space problem, redistribute work or handle very large data files are statements of resource allocation policy. The management of JES queue space in YES/MVS I is unique in the large policy component of the encoded knowledge and because of the appropriateness of its "opportunistic" approach to problem resolution.

Additional characteristics of operational knowledge. The preceding subsections have attempted to provide insight into the nature and inherent complexity of the operator's job and into the way YES/MVS attempts to imitate the actions of an expert operator. During the development and use of YES/MVS, a number of other lessons were learned or found to be important about the operator's job, about the knowledge required to operate large computing systems, and about the implications of these facts with respect to the development of facilities that would automate large-systems operation. In this subsection, we outline some of these insights, especially those that are pertinent later in the paper.

As mentioned in the introduction, an expert operator at one computing center would typically require a significant period of training to become expert at another computing center because of significant variations in

Procedure books outline how systems are to be operated.

1. Installed equipment and its interconnection and physical layout, including types of consoles.
2. Installed operating system, features, and subsystems.
3. Installed monitoring packages and their parameterization and intended use.
4. Message content, routing, and filtering.
5. Appropriate commands, keyboards, and keys with dedicated meanings.
6. Workload types—transaction processing, batch, development and test, and process control.
7. Resource allocation policy: dedicated and shared components, priorities.
8. Problem-handling techniques and problem-response policies.
9. Organization and management of operations staff, especially the subdivision of responsibilities.

Most large computing centers maintain *procedure books* that outline how the systems are to be operated in both standard and problem situations. Because of the ongoing evolution of systems, maintenance of such a document is frequently a very time-consuming task. The procedure book must be studied carefully by any new operator, even one with experience from another computing center. In some sense, it is the goal of operator automation facilities to provide a natural and reasonably simple means for completely and precisely stating the contents of a procedure book, and then for the knowledge in that procedure book to be used to control the system automatically in real time.

To reduce the difficulty in writing and maintaining such an "executable procedure book" for automatic operation, uniformity in approaches to operation must be sought and variations in operational approaches understood and minimized. There are several identifiable kinds of variation among large computing centers in operational approaches to resource allocation and problem handling. We shall use the discussion in the previous section on JES queue space problems to illustrate.

1. There are variations that are parameterizations of a common approach in the same way that an operating system is parameterized during system generation or initialization to reflect the particular hardware configuration on which it will run. The underlying strategies are uniform, but the environment to which they are to be applied is variable. In the JES queue space example, parameterizations are possible for the space dedicated to the JES queue, the detection thresholds, and identification of printers that can be used to relieve queue space.
2. There are variations in the strategies for doing things (allocating resources or handling problems) that reflect deep variations in purpose and intent. In the queue space example, the dedication of a particular printer to printing a particular kind of job might be crucial to the service being supplied by a computing center. Under essentially no circumstances would this printer be available for other kinds of work. At another center, printers might be a common resource, each providing similar services. In the latter computing center, any job could be routed to the least utilized printer. Rerouting of jobs to alleviate JES queue space problems may be customary at one site and totally unacceptable at another.
3. There are more shallow variations in strategies reflecting changes in approaches that may have evolved over time but that do not have inherently opposed goals. Large dumps and data files left on the JES queue can cause queue space problems. One computing center might have their operators check each day for material over a specific age. The policy might be to destroy the data and free the space after warning the owner and allowing him time to save the data elsewhere. At another site, the adopted policy might be to dump such data files immediately to tape whenever a queue space problem is encountered. Although there might be deep-seated motivations for the variation in policy, it is more likely that personnel at each center have evolved their own mechanism for addressing the problem and that an automatic facility could be developed to satisfy each.

The difference between the second and third categories (above) is essentially a matter of degree, so there is no clear delineation between the two. Rather, there is a spectrum of possibilities. At one end the selected policy is critical; at the other, no one has strong preferences, so long as it gets the job done.

Variations in operational policy of type 1 and type 3 in the above list appear amenable to the development of one general-purpose facility that would automate most operations in an acceptable way. However, policy variations of type 2 are widespread and establish the requirement that a facility that automates operations must be able to be readily customized to reflect local operational policy when it is installed.

Experts cannot provide a complete description of their knowledge without an iterative process of probing.

It is also instructive to note that the operator is the "court of last resort" for keeping systems in working order. When a subsystem has software that detects an extremely complex situation or a situation requiring knowledge of management policy for resolution, the developers have had little choice other than to put in code to send a message to the operator (via the WTO, or Write To Operator, facility). As resource allocation mechanisms and recovery management schemes have grown increasingly sophisticated during recent decades, operators are left with the situations that developers were unable to handle—usually because they involved policy, extreme complexity, or both.

The fact that the operator is left with the most unusual problems was reflected during YES/MVS development by the variety of software that had to be written to automate operator actions. The scheduling done by YES/MVS required the maintenance of queues of control blocks representing pieces of work. The monitoring was more characteristic of other real-time, process control applications. The diagnosis involved heuristically directed search techniques. The problem resolution frequently had a significant planning component.

At this point, it should be obvious that operators do not take significant actions based solely on a single message or timed interrupt. Instead, operators retain extensive information about the historical and recent status of the target system, and they only make decisions in the context of that status information. There are several identifiable kinds of status information that operators retain and use in making decisions:

1. Visual and temporal placement and association of messages on an operator's console.
2. System status, e.g., current system parameterization, workload, status of I/O devices.
3. Age and reliability of their own status model—Operators should know when their information is old and should be refreshed.
4. Status of resolution of a problem or of the task at hand—An operator knows what has been learned in diagnosing a problem, what actions have been taken, whether those actions worked, etc.

As has been reported in other expert-systems projects, we observed in the development of YES/MVS I that experts (especially where the expertise involves solving problems that are complex and may occur infrequently) cannot provide a complete description of their knowledge without an iterative process of probing. Indeed, it is often difficult for experts to organize their knowledge. We repeatedly encountered the following scenario: The *knowledge engineer* (programmer of the knowledge base) asks the problem-handling expert how he handles a situation.

Over a period of hours or a few days the expert provides a response. The knowledge engineer goes off to write the software that captures that knowledge; he finds holes and comes back with more questions. These questions trigger the recognition of additional patterns in the expert's memory, and the expert describes answers to the immediate questions and frequently provides additional important material. The above is iterated until it appears to the knowledge engineer that the encoded knowledge is ready for testing. Testing proceeds, problems are found, and the knowledge engineer returns to the expert, who quickly recognizes that he has omitted additional points. Thereafter the testing/additional

knowledge loop is iterated until the tests are run successfully and the facility is reasonably stable.

The value of the rule-based approach. In the course of developing YES/MVS I, we discovered that operators typically describe their knowledge in terms of situations and responses. A typical example might be

```
If there is a JES queue space problem,
  and there is a dump of more than 100 000 lines on
  the queue
                that is more than 24 hours old,
Then I send the owner mail asking whether we can
  get rid of it.
```

This observation suggests that rule-based programming is a very natural vehicle for capturing operational knowledge. The condition-action rules match the "given a situation, take an action" flavor in which an operator typically describes his knowledge.

To elaborate, consider the three major areas of responsibility for operators: monitoring, resource allocation, and problem handling. Monitoring knowledge can be conveniently expressed in the form of condition-action rules; the left-hand side specifies the circumstances under which a query is required, and the right-hand side indicates the actions necessary to perform the query. Resource allocation is also natural to express in terms of rules. Here, the left-hand side indicates when an allocation change should occur, and the right-hand side states the actions necessary to effect that change. For example,

```
if system load is not too heavy
then allocate more batch initiators
```

Lastly, problem handling nicely fits the situation-response paradigm where the situation consists of conditions about current status and past history, and the response consists of actions to improve the system. For example,

```
if JES queue space is now at an acceptable level
   and the processor was previously varied off line
            to save JES queue space
then vary the processor back on line
```

Our experience with YES/MVS I indicated that the benefits of rules go well beyond their being a good representation for operational knowledge. Rules encourage modularity since each rule contains in its LHS a complete description of prerequisites for RHS invocation and so is a self-contained "chunk" of knowledge. Developers normally seek to minimize rule interdependence and to make dependencies explicit where they are required. During the YES/MVS development process, different people have been able to work independently on different subdomains of operations partly because of the modularity of rule-based techniques.

As previously mentioned, knowledge engineering is an iterative process in which changes are repeatedly made to better conform with the expert's expectations. Thus, it is essential that during this process software techniques are used that support an "additive" process of encoding the knowledge; i.e., more knowledge can be added with minimal disruption of the previously encoded knowledge. If major reorganization of the software is required for each addition, the task will become intractable. The modularity of rules made it easy not only to add needed "chunks" of knowledge but also to delete unneeded or change incorrect ones.

Another perspective of rule-based programming is that the inference engine provides a mechanism for dispatching based on context. Recall that much of operations appears to be context-driven. For example, the LHS of the previous example looks at the current level of JES queue space, the status of the processor, and the motivation for varying the processor off line, if it is off line. An operator's actions most frequently consist of short bursts of activity (most commonly command submission and physical intervention) interleaved with pauses while awaiting messages generated by the target system or responses to commands. During pauses there usually is an enormous variety of messages or responses that could occur next.

Had we used a procedural language to automate operations, a special-purpose dispatcher would have been needed to dispatch the varied, short bursts of activity. Much of the software would consist of logic to control the context-driven selection of what to dispatch next. By using rule-based software techniques, we have a built-in facility for context-driven dispatching. Production rules provide a language for expressing complex contextual situations (in the LHSs of rules) associated with the short tasks that are appropriate in that context.

Model management. It was previously mentioned that operators maintain a conceptual model of the computing complex. Like a human operator, YES/

MVS must maintain a model of the target system, including information such as the status of devices, the age of the status information, and the current status of solutions to particular problems. In addition, since YES/MVS interacts with human operators, a model must be kept of the operators themselves, including items such as what requests have been sent to the operator and what responses the operator has made.

Automating MVS operations is not a matter of building a single expert system.

Since operational knowledge is encoded as rules and rules rely on working memory to establish a context, models of the target system and the operator must be maintained in working memory. Obtaining status information involves issuing a command (e.g., a JES inquiry), interpreting the resulting reply, and handling nonresponses. In YES/MVS I each subdomain independently solicited and maintained its own status information. Often this led to duplication of effort. For example, both the JES queue space and the hardware error experts are concerned with the operational status of printers. Even if a piece of status information is not shared among subdomains, it is still worthwhile to provide a common service for model management so that functions such as handling nonresponses are treated in a consistent manner.

Building on the YES/MVS I experience

An important lesson we learned from YES/MVS I is that automating MVS operations is not a matter of building a single expert system. Rather, due to the wide variations among computing centers and the different, ongoing changes which occur even within a single computing center, it will be necessary to provide a family of related expert systems. Thus, the focus of YES/MVS II is to provide an *expert-systems shell* which is the basis for automating operations in a family of expert systems. Our intent is that the shell (1) incorporate the knowledge and services common to expert-systems automating operations as well as (2) facilitate customizing an expert system for a particular computing center. Such an approach may greatly improve the economic feasibility of automating operations.

Knowledge representation. Our experience with YES/MVS indicates that operators most frequently express their knowledge in terms of situation-response statements. Thus, a good knowledge representation ensures that each operator situation-response statement can be represented by a single rule in the knowledge base, with the operator's situation corresponding to the left-hand side of the rule and the operator's response corresponding to the right-hand side of the rule. By so doing, operations experts and knowledge engineers can more easily write, understand, and modify encoded knowledge because the necessity to analyze the interaction between rules is minimized.

During the development of YES/MVS I in OPS5, we were often unable to achieve the goal of a one-rule-per-operator situation-response statement. Indeed, this failure was a major motivation for developing a new language for building expert systems, the Yorktown Expert Systems/Language One (YES/L1).[19] YES/L1 is a data-driven rule-based language which includes PL/I as a subset. Although YES/L1 is not discussed in detail here, we do point out some of its key features.

Foremost, we discovered that the expert-systems development language should support powerful constructs on the left-hand side. For example, the JES queue space expert maintains a symbolic variable that indicates the current severity of any JES queue space problem. This variable can take on any of five values, and the current value is called the severity mode of the current problem (when there is a problem). To avoid repeated switching between modes during borderline problems, the conditions for entry into and exit from a mode are slightly more complicated than a simple numerical threshold. Rules are needed to maintain the correct severity mode and to start/stop additional status information querying as may be needed when the mode becomes more/less severe. Using OPS5, several approaches are possible for writing these rules, but all require a number of rules. (The YES/MVS I version employed thirteen rules.) If more complex conditions could be stated in rules, for instance, comparisons involving arithmetic computations or subfunctions, this function

could be nicely written in a few rules. (Two rules seems best in YES/L1.)

The real-time nature of YES/MVS also creates special needs for expressing operator-defined situations. Specifically, a major component of YES/MVS is the maintenance of information about the target system. In YES/MVS we were forced to include on the left-hand side any WME referenced on the right-hand side (due to OPS5), which meant that the left-hand side of a rule no longer corresponded to the operator's statement of a situation. YES/L1 provides a built-in function for selecting particular WMEs on the right-hand side of a rule; thus, WMEs that are not related to an operator-defined situation can be referenced on the right-hand side without including them on the left-hand side.

Another problem related to the ongoing maintenance of information is when to re-instantiate a previously fired rule that references a WME with an attribute whose value has changed. In OPS5, a WME is considered new (and hence can re-instantiate a rule) if *any* of its attributes are changed. For example, every time the status attribute of a printer WME is changed, the old WME is destroyed and a replacement with the new value for the attribute is created. This has undesirable side effects for a rule which is only interested in the list of all IBM 3211 printers and so only refers to the type attribute of the printer WMEs; every time a change is made to the status attribute of a 3211 printer, a new printer WME is created which in turn could cause the "list-3211-printers" rule to fire. This is neither obvious nor desirable. YES/L1 avoids this difficulty by providing an attribute-level granularity for changes to WMEs. Thus, an instantiation re-enters the conflict set only if there is a change to an attribute referenced by the rule.

We have found that adequately representing the action portion of operational knowledge requires expressing procedural behavior such as iteration and if-then-else constructs. For example, when the JES queue space expert analyzes the printer queues, it iterates over all printers, and for each printer it iterates over the set of jobs waiting so as to compute the number of lines waiting to print. Such actions cannot be expressed on the right-hand side of a single OPS5 rule, since only sequential code is permitted. However, in YES/L1 the right-hand side can employ any PL/I procedural constructs.

We also found that the actions themselves may be embodied in sets of rules. For example, the JES queue space expert has a hierarchy of rules consisting of (1) determining the problem severity mode (e.g., if JES queue space is very low, consider panic-mode actions), (2) heuristics to try once the severity is known (e.g., if a shared printer is not reserved by the target system, reserve that printer), and (3) steps within each heuristic (e.g., if the printer status is unknown, query the target system). In OPS5, rule hierarchies are not easy to implement, but they can be concocted using a variety of flag attributes. YES/L1 will provide rule hierarchies through a rule subroutine construct that permits invoking a set of rules from the right-hand side of another rule.

Appropriate action in some problem situations should depend on the availability and convenience of the operator.

Model manager. In retrospect, we recognize that managing status models of the external environment is a key component of a YES/MVS expert. The YES/MVS expert must have current status information at all times in order to perform monitoring, resource allocation, and problem handling. The absence of a built-in model manager in YES/MVS 1 led to several problems:

1. Separate queries were issued by each YES/MVS expert for possibly the same information, and each expert was separately responsible for handling nonresponses.
2. It was difficult to share information among experts since the information was in different classes of working memory. Hence, information was duplicated, which could cause inconsistencies.
3. There was no central point of control over query policies, such as how frequently the target should be prompted.
4. In complex problem situations it can be useful to prioritize queries to be issued either to the target system or to the operator. When queries are diffused throughout several subdomain experts,

there is no easy mechanism for establishing relative priorities.

To alleviate these problems, we have included model management as a system-supplied service in YES/MVS II. The YES/MVS model manager is a rule-based program with responsibility for maintaining in working memory information about the state of the target system and of interactions with the operator. The central points concerning the model manager are as follows:

1. The model manager defines and provides a *standard* representation for status information. When designed with care and foresight, the format of this information can remain largely unchanged across sites and changes in MVS and JES.
2. So far as is reasonable, the model manager *automatically* maintains the needed, current status information.

By depending on a standard and current model of the target system, the code of the expert can enjoy a cleaner view of the target system, a view at a higher logical level that is free from the details of query and response formats, message and command timings, and message variations. The model manager has built-in facilities for automatically querying the target system. Responses of the target system to these queries are processed by the model manager and are stored in standard WMEs. These WMEs are then referenced in the LHSs of rules that control knowledge-based actions.

In order to eliminate unnecessary overhead in maintaining the status model of the target system, expert code for each interested subdomain may express its requirement for or noninterest in a particular class of status information. If no subdomain cares, regular queries may be suspended for that information. Expert code may also make special requests asking that particular status information be refreshed or may mark particular status items as "old" and in need of being refreshed.

Query blocks (blocks of information about each type of query and its response) standardize and control the services provided by the target system model manager. This approach is intended to reduce the work required to customize and maintain the target system model. General strategies are also employed for identifying and responding to nonresponses and unusual circumstances in communicating with the target system.

There are also benefits that accrue from the existence of a standard and current model of interactions with the operator. Software to control the presentation of information becomes more uniform and thus easier to read and write. Also, care must be taken not to flood the operator with requests. The appropriate action in some problem situations should depend on the availability and convenience of the operator. With a standard, generally available model of the operator, honoring such considerations becomes reasonable.

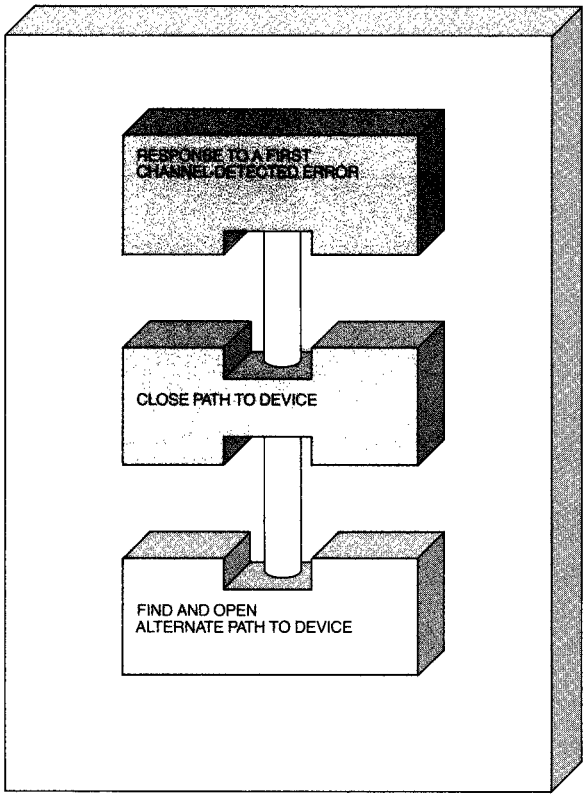
Generally written knowledge. For some types of operational problems, there are generally applicable strategies for problem handling. These strategies could be uniformly applied at almost all computing centers. As part of the effort to develop the base for a family of related expert systems, each reflecting local operational policy, such general strategies should be encoded once and then parameterized to be specific to each computing center. The required parameterization may be extensive, including a description of the configuration, specification of thresholds, etc., but it appears that for some problem domains the approach is quite reasonable. Here, we consider the response to certain hardware failures in a system as a specific example.

If a hardware channel detects and reports to MVS a hardware failure (a Channel-Detected Error, or CDE) while trying to execute a read or write operation to a device, e.g., to a Direct Access Storage Device (DASD), the error-recovery mechanism in MVS automatically retries the operation. If the error persists, MVS notifies the operator. Once notified, the operator assumes (at least in an MVS Extended Architecture system) that the error is not transient. Thus, something should be amputated from the system to reduce the likelihood of the problem cascading and bringing down the entire system. Explicit information is not normally known about the exact source of the problem, and decisions must be made with the information available.

When operators were interviewed about how they handle hardware errors, they responded in terms of specific actions they perform for specific situations. Only with substantial thought and comments from multiple human experts did we develop the general strategies listed below:

1. Consider amputation only for nonessential components.
2. Maintain open paths to devices whenever possible.

Figure 2 Decision tree for strategy for handling first channel-detected error



3. When faced with an error on a path, assume that the least critical component on the path is the failing component.
4. When faced with multiple failing paths, assume that the failing component is at the least critical point of intersection of all the failing paths.

These general strategies can be recast into general-purpose productions that can control the response to hardware errors provided that they have available information on the following points:

1. All hardware devices in the configuration.
2. All possible paths to devices.
3. What devices are essential to keep the system operational.
4. What paths are currently open in the system.

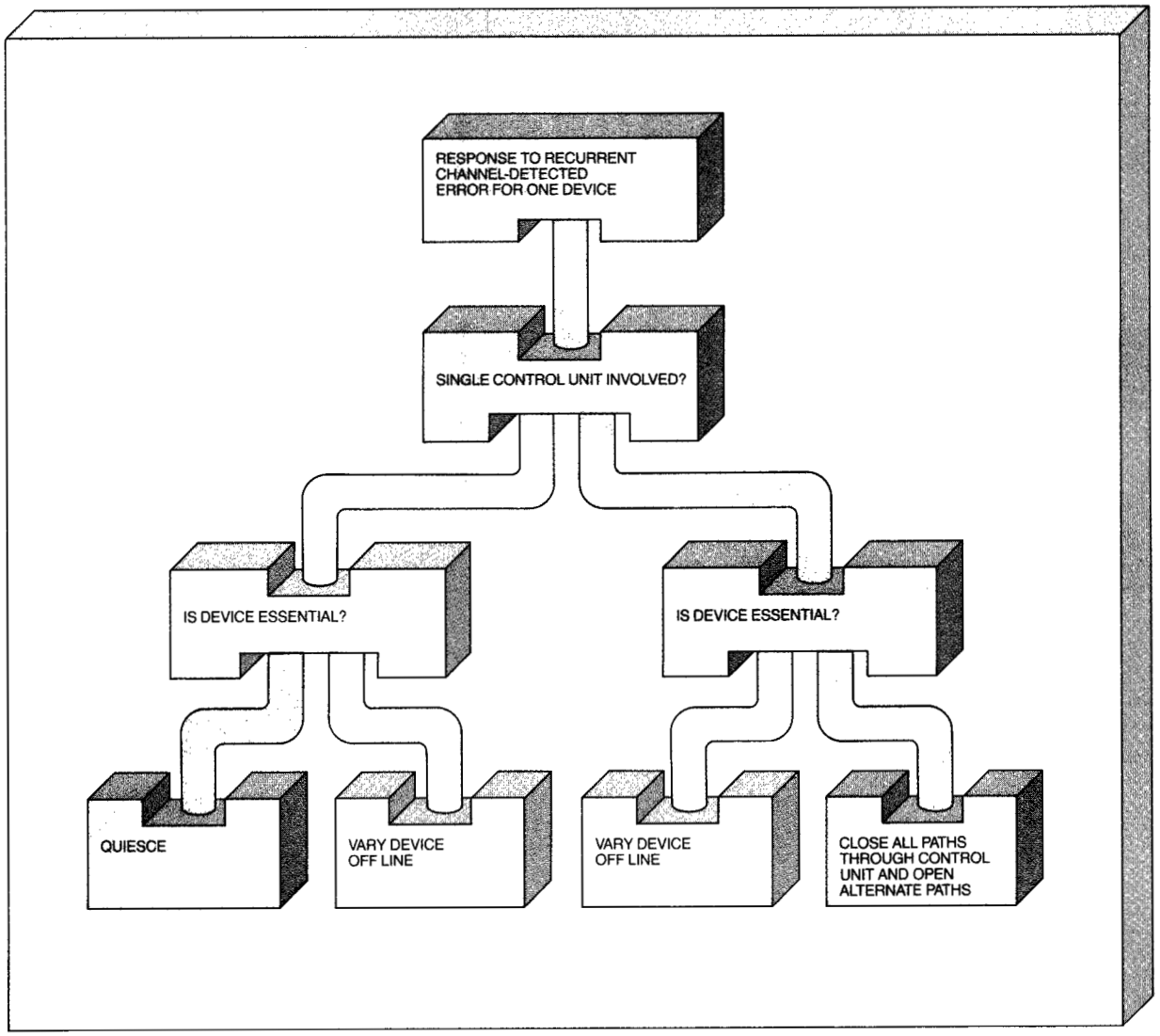
Lists of all devices and all possible paths are specified to the MVS operating system at system generation time. These items are provided to YES/MVS at initialization. A list of the essential devices must frequently be made especially for this purpose. Information on what paths are currently open may be obtained by submitting queries to the operating system or can be continuously maintained by keeping track of all path status changes reported by the operating system.

The general strategy (slightly simplified) for diagnosing and responding to channel-detected errors can then be described in the following productions:

```
if a device is associated with a channel-detected
error
then if the device is essential to the system
     then assume there is a path problem
                between the device and control
                unit
     else (nonessential device) conclude the device
     is failing

if there are multiple errors on a device
                involving different control units
then conclude that the device is failing

if there are multiple errors on a device
                involving just one control unit
then if device is essential
     then assume the control unit is failing
     else (nonessential device) conclude the device
     is failing

if there are errors on different devices attached
                to the same control unit
then conclude that the control unit is failing

if a device is failing
then vary the device off line

if there is a path problem between a device and a
control unit
then close the path, and
     if an alternate path can be found
     then open an alternate path to the device

if there is a failing control unit
then close all paths through the control unit, and
     for each affected device (if possible) open an
     alternate path
```

The above productions, in fact, encode the general strategy used in treating a significant number of hardware errors. The first four productions diagnose the cause of a hardware problem. The last three productions can be generalized to apply a standard

Figure 3 Decision tree for strategy for handling recurrent channel-detected error



treatment that varies with the type of problem and the failing component. The productions use configuration information to determine whether alternate paths are available and to locate the intersection of multiple errors. The classification of devices as essential or nonessential is used to focus the diagnosis on treatable problems.

The problem-solving principles in this domain can be succinctly stated and can be represented in other forms besides productions. For example, decision trees can be used to state clearly what actions are to be taken under what circumstances. Decision trees for handling CDEs appear in Figures 2 and 3. Most people find the decision trees the simplest format for quickly gaining insight into the diagnosis and handling of CDEs. It should be emphasized, however, that knowledge engineers only arrived at the decision-tree format after carefully analyzing the knowledge approved by experts on the handling of hardware errors. Decision-tree representations of the appropriate response to problems were found for other problem areas also after substantial analysis by the knowledge engineer.

Testing techniques. Since YES/MVS actually controls portions of a computing complex, it is imperative that YES/MVS operate with minimal errors. Doing so requires thorough and effective testing. Testing is needed not only for the standard portions of YES/MVS, but also for customization done at individual sites.

One approach to testing is to have YES/MVS actually operate an MVS system. Although this form of testing must always be done, by itself it has severe shortcomings. First, if testing is done on a dedicated test system, a meaningful workload will not be present. Alternatively, YES/MVS (running in advisory mode) can be tested on a system in production use. YES/MVS II is organized to allow multiple experts to tap into the message stream of the system. This arrangement permits on-line testing (in advisory mode) of changes to the operational knowledge base. However, it is still difficult to provide thorough testing, since on a real production cluster, the problems with which YES/MVS deals are (hopefully) rare.

Another approach to testing YES/MVS is based on a library of scripts that could be replayed to test that YES/MVS is responding correctly to previously encountered problem situations. The replaying of scripts would be used primarily to test advisory-mode actions of YES/MVS, as the script would simulate changes over time in the target system but would not contain sufficient complexity to reflect responses to active commands. The messages and responses would be extracted (with timing information) from logs of previously encountered, real situations. When replayed, a script would simulate the generation of asynchronous messages. Simulated responses to queries for status information would be based on special *time events* manually inserted into a script. Each time event would control, on the basis of timing, the response of the script to a specific operator query. There should be a library of standard responses that could be inserted. The expected advisory output from YES/MVS would be associated with the script and would be compared with the actual output. When errors in YES/MVS II were found and repaired, a script would be created to test for the correct response. Over time, a library of scripts should be available to provide a reasonably comprehensive test of the advisory-mode function of YES/MVS.

Testing of YES/MVS active-mode function requires a more sophisticated simulation of the response of a target system to active commands. In YES/MVS I, the scheduler of large batch jobs was tested extensively in this manner. Such simulators would simulate nearly all the externally visible states of the target but relatively few of the internal states, and would accurately simulate the target state transitions caused by active expert commands but only crudely simulate the target state transitions caused by the production workload.

Effective testing is crucial to the success of YES/MVS.

Simulation tests will be able to stress YES/MVS in ways that are difficult or impossible with a real system, such as inducing rare hardware failures and creating very high message rates. A shortcoming of this approach is that one tests only how one *thinks* the target system behaves; many errors result from the target system deviating from our expectations.

In conclusion, effective testing is crucial to the success of YES/MVS, but no single testing technique is all-encompassing. We anticipate that all of the aforementioned techniques will be used in testing YES/MVS II.

Knowledge acquisition for automating operations. Each computing center needs to have its own expert system for operations because of the variations in operational policy that exist among computing centers. The strategy employed by YES/MVS II is to simplify the development of such expert systems by providing a standard knowledge base which can be easily customized to reflect the unique characteristics of individual computer centers. Tools to aid in this customization process would be of significant value.

General-purpose knowledge-acquisition tools have proven difficult to develop because knowledge acquisition itself is an activity requiring large amounts of knowledge. However, developing valuable *special-purpose* knowledge-acquisition tools should be much easier. Knowledge of the domain can be built into

the tool; outlines of typical statements of domain knowledge can be included for each of the various kinds of knowledge that can be anticipated; reasonableness checking can be provided; tools to assist with testing can be included; and standard libraries of cases can be developed to facilitate testing standard features. The authors are enthusiastic about the potential value of knowledge-acquisition tools that assist with the process of customizing a standard knowledge base to reflect the unique policy of a computing center, and work has begun on the development of such a facility. We outline here some of the observations that have been guideposts in our design efforts.

Remember that when the knowledge-acquisition process was begun for YES/MVS, the available operational knowledge was predominantly in the form of situation-response statements. The rule-based paradigm proved to be a natural knowledge representation. Decision trees would be another organization for representing the knowledge of diagnosing and resolving operational problems. Decision trees may or may not be a friendly representation, but in a theoretical sense, decision trees must exist. After knowledge engineers had worked on the various system operation problems for a period of time, decision trees and general principles for handling particular problems began to appear. Examples were described in the subsection on generally written knowledge for operator response to hardware failures.

Decision trees provide a hierarchical decomposition of problems, but other, more flexible, decompositions seem more easily applied to the broad class of system operation problems. Consider a hierarchical decomposition of a problem state into increasingly specific substates where each substate has the following items associated with it:

1. A plan for how to extricate the system from this particular state—The plan could include primitive corrective actions from a library of actions, and the enabling and disabling of entry into substates.
2. Conditions for entry into the state.
3. Conditions for exit from the state either because of resolution of the problem condition or because a change in status makes this substate inappropriate.
4. Description of the status information needed to be able to determine entry into and exit from this state.

These hierarchical decompositions differ from decision trees in several material ways. First, control can be simultaneously passed to multiple substates of a particular state; i.e., substates need not be mutually exclusive. Second, entry into a substate is not based solely on a simple decision but can be controlled by the plan associated with a state for extrication from the state.

Our preliminary experience indicates that when such a decomposition for an operational problem is known, first, the decomposition is fairly easy to understand and second, identifying and making needed modifications to the problem solution strategy are simpler than they would be with other knowledge representations we have considered. We intend to develop a compiler that would generate YES/L1 code from the hierarchical decomposition representation of the problem-solving strategy. The potential advantages of this representation are as follows:

1. With the use of graphics and the ability to zoom in on nodes (substates) in the hierarchical decomposition, it appears that it will be easy to convey an overview of the strategy for handling the associated problem.
2. Having an overview of the strategy should reduce the difficulty of determining the global impact of local software changes.
3. Problems are outlined as though all needed status information were available and current. Thus, issues of the (non)availability of status information and the suspension of execution while awaiting responses to queries and active commands are compartmentalized and hidden in an overview of the problem-handling strategy.
4. The knowledge engineer will be able to focus on the strategy for problem resolution, and the compiler will introduce dependencies (on the built-in model manager) for collection of status information.
5. The compiler could supply reasonableness checking where possible.

It should be emphasized that the facility would be intended primarily for use in customizing rather than in developing an operational knowledge base. The user of the knowledge-acquisition facility would start with descriptions of standard (and hopefully comprehensive) responses to most operational problems. The user would prune, reparameterize, and make (hopefully minor) modifications to represent the special characteristics of the policy of one computing center.

Conclusions

Operators have different responsibilities in different computing centers, but a few generalities seem to be valid. Twenty years ago, operators provided both problem-handling expertise and control over the normal allocation of computing resources. Since that time, the allocation of computing resources has largely been automated, but the problem-handling abilities of large-systems operators have been only partially automated. This is because of (1) the complexity of detecting, diagnosing, and responding to the range of unusual but possible system problems and (2) the variation in operational policy among computing centers that make it difficult to select a single "correct" scheme for responding to operational problems.

Several software techniques came out of the YES/MVS I effort. But for those interested in automating large-systems operations, the most important lesson learned from our experience with YES/MVS I was that it is feasible to automate the complex decision-making processes operators use to detect, diagnose, and respond to problems in large computing complexes. The effort to develop YES/MVS I involved many tasks besides knowledge engineering. Still, the problem-handling rules in YES/MVS I required a number of man-years of effort and even then were certainly not complete. Thus the expense would be very high if every large computing center had to develop its own problem-handling rule base.

Although YES/MVS I established that large-systems problem handling could be automated, the economic viability of automated problem handling remained unproven. This is the reason why the design of YES/MVS II puts such great emphasis on software techniques that will ease the customization of the facility when it is installed at a new computing center and that will ease ongoing maintenance to reflect changes in configuration, workload, and operational policy.

This paper has concentrated on those aspects of our experience with YES/MVS I that appear to be significant if a knowledge base for large-systems operation is to be made relatively easy to customize and maintain. On the basis of this experience, it appears that leverage can be obtained from the following:

1. The characteristics that are common to most operational problems.
2. For each type of problem, the characteristics that are common to almost all solutions of that problem.

These considerations have led us to consider several different approaches to the customization and maintenance problems. Among them are the following:

1. Enhancing the development environment (languages, debugging, and testing tools) available for encoding and refining a knowledge base for large-systems operation.
2. Attempting to increase uniformity in approaches to problem handling so that greater portions of the problem-handling knowledge can be organized as "system-supplied" services.
3. Seeking the most general approaches to specific types of problems, so that only parameterization or minimal customization is required to reflect the unique characteristics of a particular computing center.
4. Exploiting uniformity across types of operational problems to develop a knowledge-acquisition tool intended to assist the user in customizing the knowledge of how to respond to a particular type of operational problem.

Though our experience with customizing YES/MVS II for installation at a new computing center remains limited, we are optimistic about the prospects for making the customization process manageable. Ultimately, we hope that the staff of a large computing center could, with reasonable effort, encode those aspects of their operational policy that are unique and could thereafter enjoy fast, consistent, and hopefully expert response to operational problems. This should, in turn, provide improved availability and performance to the users of the computing resource.

Acknowledgments

YES/MVS I was developed as a joint effort by the Expert Systems for Systems Management group and the Expert Systems group at IBM's Thomas J. Watson Research Center. The Expert Systems group has embarked on other projects since the completion of YES/MVS I, but several members of that group should be acknowledged here for their contributions to YES/MVS I, including J. H. Griesmer, S. J. Hong, M. Karnaugh, J. K. Kastner, and M. I. Schor.

Cited references

1. E. H. Shortliffe, *Computer-Based Medical Consultation: MYCIN*, American Elsevier, New York (1976).
2. R. G. Smith and J. D. Baker, "The DIPMETER advisor system," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence* (1983), pp. 122–129.

3. R. O. Duda, J. G. Gaschnig, and P. E. Hart, "Model design of the PROSPECTOR consultant system for mineral exploration," in *Expert Systems in the Micro-Electronic Age*, D. Michie (editor), Edinburgh University Press, Edinburgh (1979), pp. 153–167.
4. Robert K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg, *Application of Artificial Intelligence for Organic Chemistry. The DENDRAL Project*, McGraw-Hill Book Co., Inc., New York (1980).
5. *MVS/Extended Architecture Overview*, GC28-1348, IBM Corporation; available through IBM branch offices.
6. *OS/VS2 MVS/System Product—JES3 General Information*, GC28-1025, IBM Corporation; available through IBM branch offices.
7. *Introduction to JES3*, GC28-0607, IBM Corporation; available through IBM branch offices.
8. R. L. Ennis, J. H. Griesmer, S. J. Hong, M. Karnaugh, J. K. Kastner, D. A. Klein, K. R. Milliken, M. I. Schor, and H. M. Van Woerkom, "YES/MVS: A continuous real-time expert system," *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX (August 1984), pp. 130–175.
9. R. L. Ennis, J. H. Griesmer, S. J. Hong, M. Karnaugh, J. K. Kastner, D. A. Klein, K. R. Milliken, M. I. Schor, and H. M. Van Woerkom, "Automation of MVS operation, an expert systems approach," *Proceedings of Computer Measurement Group Conference XV*, San Francisco, CA (December 1984).
10. R. L. Ennis, J. H. Griesmer, S. J. Hong, M. Karnaugh, J. K. Kastner, D. A. Klein, K. R. Milliken, M. I. Schor, and H. M. Van Woerkom, "A continuous real-time expert system for computer operations," *IBM Journal of Research and Development* **30,** No. 1, 14–28 (January 1986).
11. A. A. Guido, "Unattended automated DP center operation: Is it achievable?" *European GUIDE Proceedings*, Lyon, France (June 7–10, 1983), pp. 440–446.
12. *VM/SP Introduction*, GC19-6200, IBM Corporation; available through IBM branch offices.
13. *LISP/VM User's Guide*, SH20-6477, IBM Corporation; available through IBM branch offices.
14. *VM/SP System Product Interpreter Reference*, SC24-5239, IBM Corporation; available through IBM branch offices.
15. L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley Publishing Co., Reading, MA, in press.
16. C. L. Forgy, *OPS5 User's Manual*, CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (July 1981).
17. C. L. Forgy, "RETE: A fast algorithm for the many pattern/many object pattern match problem," *Artificial Intelligence* **19,** No. 1, 17–37 (September 1982).
18. M. I. Schor, "Declarative knowledge programming: Better than procedural?" *IEEE Expert* **1,** No. 1, 36–43 (Spring 1986).
19. K. R. Milliken, A. V. Cruise, R. L. Ennis, J. L. Hellerstein, M. J. Masullo, M. Rosenbloom, and H. M. Van Woerkom, *YES/L1: A Language for Implementing Real-Time Expert Systems*, Research Report RC-11500, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (December 1985).

Keith R. Milliken *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Milliken is a Research staff member and manager of "Systems Management" research and of the "Expert Systems for Systems Management" project. He has been involved for the last several years predominantly with the development of experimental expert systems to assist with the management of large computing systems. Dr. Milliken received a bachelor's degree in mathematics from Occidental College in 1968 and a Ph.D. in mathematics from the University of California at Los Angeles in 1975. During 1976 he was a Member of the Institute for Advanced Study in Princeton, New Jersey. From 1975 to 1979 he was an Assistant Professor of Mathematics at California Polytechnic State University. He joined IBM in 1979. His interests include expert systems, the management of large computing systems, and combinatorial mathematics and algorithms. He is the author of research papers involving each of these areas.

Anthony V. Cruise *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Cruise is an advisory programmer and a member of the "Expert System Shells for Systems Management" project. He is primarily involved with compiler construction, in particular for the YES/L1 compiler. Before joining IBM at the Research Center in 1984, Mr. Cruise worked in compiler construction and systems and applications programming. He is coauthor of a previous paper on the expert-systems language YES/L1.

Robert L. Ennis *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Ennis is a programmer at the Research Center and a member of the "Expert System Shells for Systems Management" project. He is interested in automated operations. Mr. Ennis joined IBM in 1966 as a computer operator and from 1970 to 1983 filled many operations management positions. In addition to his management duties, he worked as a programmer on the SEM Project, a Series/1-based environment monitor.

Allan J. Finkel *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Finkel is a Research staff member and a member of the "Expert Systems for Systems Management" project. Dr. Finkel received a bachelor's degree in mathematics from the State University of New York at Binghamton in 1977 and a Ph.D in mathematics in 1982 from New York University, where he studied at the Courant Institute of Mathematical Sciences. During 1982–1983 he was a Member of the Institute for Advanced Study in Princeton, New Jersey. He joined the Mathematical Sciences Department of IBM Research in 1983 as a Postdoctoral Fellow and moved to the project group in 1985. His interests include expert systems, the management of large computing centers, and differential equations.

Joseph L. Hellerstein *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Hellerstein is a Research staff member. He received the B.A. in mathematics from the University of Michigan, Ann Arbor, in 1974, and the M.S. and Ph.D. in computer science from the University of California at Los Angeles in 1979 and 1984. From 1974 to 1978, he developed communication, networking, and operating-systems software for Honeywell Information Systems. His current research interests include artificial intelligence, performance modeling, performance tuning, and expert systems.

David J. Loeb *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Loeb did his undergraduate work at the University of Illinois.

He is continuing his studies at Rutgers University while in a work-study program with the "Expert System Shells for Systems Management" project. His current research interests include machine learning, pattern matching, and knowledge representation.

David A. Klein *Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania 19104.* Mr. Klein is a Ph.D. candidate in the Department of Computer and Information Science at the University of Pennsylvania, where he holds an IBM Graduate Fellowship. He received an M.B.A. (with distinction) in decision sciences from the Wharton School and an M.S.E. in computer and information science from the University of Pennsylvania. Mr. Klein has held positions as a visiting scientist with the IBM Research Division and as a consultant for American Management Systems.

Miriam J. Masullo *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mrs. Masullo is an advisory programmer at the Research Center and a member of the "Expert System Shells for Systems Management" project. She has been predominantly concerned with the design and construction of improved facilities for the person-machine interface. Mrs. Masullo received a bachelor's degree in 1973 and a master of science degree in computer science in 1984 from the City College of the City University of New York. She is currently a Ph.D. candidate at the Graduate Center of CUNY working in the area of artificial intelligence. Prior to joining IBM at the Research Center in 1985, she had worked for more than ten years in MVS systems programming and programming support. Her research interests are in the development of artificial intelligence techniques for application to the large-systems environment and in developments at the interface between natural-language processing and data base theory.

Hugo M. Van Woerkom *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Van Woerkom is an advisory programmer at the Research Center and manager of the "Expert System Shells for Systems Management" project, which is concerned with building expert-systems shells that assist in the automation of large computing systems. He joined IBM in 1966 in Kingston, New York, where he worked in diagnostic, scientific, and systems programming. He has been a member of the Research Division since 1983. Mr. Van Woerkom received a B.A. in philosophy from Carroll College in 1961, an M.Div. from St. Thomas Seminary in 1965, and an M.S. in computer science from Syracuse University in 1984. His interests include expert systems and the control and management of large computing systems and networks. He is a coauthor of research papers in the areas of expert systems and installation management.

Norman B. Waite *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Waite is a Research staff member and a member of the "Expert Systems for Systems Management" project. He is primarily involved with the organization of a knowledge base for the automation of the operation of MVS. Dr. Waite received a B.S. with honors in mathematics in 1964 from Rhodes College (formerly Southwestern at Memphis), Memphis, Tennessee, an M.S.E. in mathematical sciences, 1977, and a Ph.D. in systems analysis, 1980, both from The Johns Hopkins University. Before graduate school, he worked in applied mathematics and scientific programming, mostly for military problems, and, during 1973–1974, was Director of Operations Research at Federal Express Corporation. After graduate school, he served on the faculty of the business school at Ohio State University and participated in a management consulting partnership. He joined IBM at the Research Center in 1985. His research interests are in expert systems for real-time process control, stochastic processes, and optimization.

Reprint Order No. G321-5269.