A continuous real-time expert system for computer operations

by R. L. Ennis
J. H. Griesmer
S. J. Hong
M. Karnaugh
J. K. Kastner
D. A. Klein
K. R. Milliken
M. I. Schor
H. M. Van Woerkom

The Yorktown Expert System/MVS Manager (or YES/MVS for short) is a continuous real-time expert system that exerts active control over a computing system and provides advice to computer operators. YES/MVS provides advice on routine operations and detects, diagnoses, and responds to problems in the computer operator's domain. This paper discusses the YES/MVS system, its domain of application, and issues that arise in the design and development of an expert system that runs continuously in real time.

1. Introduction

The requirement for high availability and high performance in large computing installations has increased the demand for fast and consistent response to operational problems. While automatic aids for computer operators are needed, such aids would be very difficult to implement and maintain in procedural software, because the operations environment is characterized by high complexity and continuous evolution. The use of expert system technology provides a basis for the implementation of operator aids that is capable of accommodating the complexity of large system operation and problem handling, yet is very flexible and well suited to evolving with the changes in an installation's operations and policies.

YES/MVS* (Yorktown Expert System/MVS Manager) was designed to assist computer operators in carrying out their activities. It is an experimental facility developed to aid in the real-time operation of a large Multiple Virtual Storage/System Product—Job Entry Subsystem 3 (MVS/SP–JES3) [1] computing installation. We henceforth refer to MVS/SP–JES3 as the *target system*. YES/MVS receives messages that would normally appear at an operator's console. It submits queries and analyzes replies to ascertain the status of the target system. Based on that analysis, YES/MVS submits active commands to MVS and its subsystems. In solving problems, YES/MVS also displays advice for the operator and indicates the manual tasks that need to be performed.

YES/MVS addresses both routine actions taken in operating the target system and spontaneous problems that would normally be handled by an operator. The problems addressed include various kinds of MVS-detected hardware errors, JES queue space depletion, channel-to-channel link problems, and subsystem abnormal ends. Problems detected and positive corrective actions are displayed for the operator, summary reports of incidents are generated, and individual incidents are automatically reported to the appropriate systems programmers via an electronic mail facility.

* A preliminary and abbreviated description of YES/MVS was presented at the 1984 National Conference on Artificial Intelligence and appears on pages 130–136 of the conference proceedings under the title "YES/MVS: A Continuous Real Time Expert System."

YES/MVS employs the techniques of *expert systems* or *knowledge-based systems* in its implementation. The problem-solving expertise of operators and systems programmers is encoded in a *knowledge base* using the high-level, declarative format of *production rules*. This knowledge base serves as a central repository for operational procedures, providing a tool for formulating and enforcing installation management policy. Since the knowledge base is inherently modular, it is easier for it to adapt and evolve with the installation than would be the case with a procedural encoding of the knowledge.

YES/MVS is among a few expert systems that execute continuously in real time (e.g., [2–4]) and that can actively exert control over the subject being monitored.

This paper describes the design of YES/MVS and reports the experience gained in developing and using YES/MVS, with emphasis on design decisions and their motivations. In Section 2, the problem domain of computer operations is characterized, and in Section 3, the specific subdomains chosen for early implementation in YES/MVS are described. Section 4 contains a brief introduction to the OPS5 production system language which was used in the development of YES/MVS. In Section 5 the organization of YES/MVS is given, and in Section 6, a specific subdomain of expertise is described in detail. The extensions to OPS5 which provided the real-time control capabilities needed in YES/MVS are discussed in Section 7. Experience in developing, testing, operating, and evaluating YES/MVS is given in Sections 8 and 9. Section 10 contains a brief history of the project, and Section 11 gives some conclusions reached concerning the use of expert systems for developing productivity aids for computer operators.

2. Problem characterization

The computer operations environment is characterized by high and usually increasing complexity. Computer operators perform many routine tasks, including mounting tapes, loading and changing forms in printers, answering phones, and actively monitoring the condition of the target system. Operators additionally watch a number of consoles for a variety of messages that may be volunteered by the target system, responding to problems as they arise. Problem resolution usually involves requesting more informational messages, consulting system documentation, submitting corrective commands, or calling for outside help. Informational message rates are often high and supporting documentation is usually voluminous. In some cases, fast operator reaction is required and there is little opportunity to consult reference material or obtain outside help.

Because the operations environment is complex and dynamic, a long training period is usually required to produce a skilled operator at a given installation. Even the training of newly hired, experienced operators can represent a significant cost to the installation, since operators' responsibilities vary with management policy, with the particular system configuration installed at a site, and with workload characteristics. The problem of operator turnover is exacerbated by the common practice of promoting experienced operators to other assignments, such as that of systems programmer.

Thus, the installation is an evolving entity with regard to both the technical environment and its personnel. Operational procedures must evolve along with the environment. Formal descriptions of operational procedures usually exist but tend to lag behind. Consequently, operators experiment and develop *ad hoc* solutions to operational problems. This results in an approach to problem solving that varies from operator to operator and increases the installation's dependence on specific personnel. Particularly effective solutions are discussed among the operators and become part of the installation's "operational folklore." Unlike formal procedures, these operational rules of thumb are not documented and distributed, and thus are not performed consistently. Consequently, they cannot be subjected to periodic management review, nor are they immediately intelligible to systems programmers who later examine the effects of operator actions in addressing system problems.

The shortage of skilled operators, the increasing complexity and speed requirements of the operator's job, and the dynamic nature of the installation call for more powerful installation management tools. In particular, tools are needed to ease the workload of the operator, to provide fast, consistent reactions to installation problems, to decrease the installation's dependence on specific personnel, to provide a basis for enforcing installation management policies, and to provide a facility for integrating new policies with old ones and testing their relative effectiveness.

An often-asked question is "Why aren't some of the modifications being built into MVS/SP–JES3 itself, rather than building an external facility such as YES/MVS?" MVS/SP–JES3 represents a product into which considerable effort has been expended in implementing automatic recovery procedures. However, it is not economically feasible to endow MVS/SP–JES3 with the capability of responding to every possible operational difficulty, particularly since each site will have a different configuration, workload, and operation policy. It is that remaining area that becomes the operator's domain of concern. Expert systems technology offers the promise of raising the threshold of operator responsibility for problem response and resolution more easily than would be possible by making modifications to the operating system itself.

3. YES/MVS application domain

Whereas YES/MVS was developed for a specific target computer, its overall objectives correspond to those outlined in the preceding section: to provide fast, accurate, and

consistent responses, both in routine situations and in problem situations, and to reorganize and reduce the message traffic that an operator must deal with in the course of his or her activities. The flexibility offered by the use of expert system techniques allows for accommodating changes in the computing environment at the IBM Research Center at Yorktown Heights, NY, and for potential future use at other installations.

At the outset, it was decided which operator activities should be included in YES/MVS in order to provide a sufficient test of the benefits expected from the use of expert systems techniques. The following subdomains were selected for implementation, since they touched a majority of those operator activities at Yorktown which involve no physical intervention.

- *Scheduling large batch jobs off prime shift*

Large batch jobs must be scheduled to balance considerations of system throughput and user satisfaction. These considerations may vary in detail from one installation to another. They include ensuring that no jobs are indefinitely delayed, employing round-robin scheduling among users submitting multiple jobs, giving priority to users who are waiting on site or who require some other special consideration, running longer jobs early in the shift, and running only those jobs that can be finished before a scheduled shutdown. Since new jobs may arrive or be withdrawn during the shift, initial scheduling may have to be changed among the jobs that are still in the queue. In Section 6, this particular subdomain is described in more detail.

- *JES queue space management*

All jobs processed under MVS are staged from a central spool file, called the Job Entry Subsystem (JES) queue space, before, during, and after execution. The operator is concerned with the remaining available queue space, because the Job Entry Subsystem cannot recover if queue space is exhausted. When the level of remaining queue space becomes critically low, many actions should be initiated to free additional space, such as altering the printer utilization policy to allow the printing of jobs which have been held, and dumping large print jobs to tape. In extreme cases, the system should be made to refuse new jobs and stop data being transmitted from other systems. To initiate such actions the operator makes use of the available facilities connected to the target MVS system. This means the operator has to perform some anticipatory actions (e.g., mounting a tape to dump jobs) as queue space decreases, and before it becomes critical.

- *Problems in channel-to-channel links*

The networking of computers at the same site is often implemented by means of I/O channel-to-channel transmission 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. Monitoring and corrective actions include periodic querying of the states of these links, using heuristics to infer line degradation, attempting to restart the links, freeing links from troublesome jobs, and rerouting the data through other computers.

- *MVS-detected hardware errors*

When MVS fails to recover from a detected hardware error, the system notifies the operator so that he or she may attempt to solve the problem. Due to the time criticality of possible remedies (such as speedy reconfiguration), otherwise-recoverable situations may result in a system crash since a human operator cannot respond in time. Responses to the most frequent hardware problems have been implemented in rules. These rules are not tied to a particular hardware configuration but rather make use of hardware configuration data placed in YES/MVS. The hardware configuration data are initially loaded from the files used in the MVS system generation process.

- *SMF management*

The System Measurement Facility (SMF) provides access to information on resource utilization in MVS. There are several routine actions that must be taken to switch SMF buffers and manage SMF data for accounting purposes. Some actions are triggered by MVS events, others are regularly scheduled.

- *Quiesce and Initial Program Load*

Before a planned shutdown the target system must be "quiesced." Routine quiescing and restarting (Initial Program Load or IPL) of MVS systems are done to test new versions of software, to install new hardware, and to allow time for maintenance procedures. The quiesce operation typically takes approximately 30 minutes and involves many operator actions.

Currently, YES/MVS does not have access to the system console. Hence, it cannot trigger a system IPL. However, YES/MVS does give advice on system IPLs, and it can take over the IPL activities once the connection to the system console becomes functional.

- *Performance monitoring*

This task goes beyond the usual scope of an operator's activities. A short-term goal is to interpret the data from existing performance monitoring software and automatically detect and classify performance problems in real time, generating summary reports in hard copy as well as in computer graphics. This work continues, and a goal is eventually to diagnose the cause of performance problems and to take corrective actions.

• *Background monitor*

In addition to the above subdomains, a set of functions in a "background monitor" was implemented. The background monitor periodically verifies that all parts of YES/MVS are operational and issues incident reports on YES/MVS activities, both routine and problem responses. Summary reports of incidents encountered during a shift or day are automatically generated. Using an electronic mail facility, descriptions of each incident are sent to the appropriate systems programmers and to the YES/MVS developers. This incident-reporting facility has proven very useful for alerting responsible personnel to system problems and for recording when problem handling code in YES/MVS was exercised. These incident reports were analyzed to aid in the further development of the YES/MVS knowledge base.

4. OPS5

As a vehicle for implementing YES/MVS, a production system language, OPS5 [5, 6], was selected. OPS5 is one of a family of expert system shells (languages with inference mechanisms) developed at Carnegie-Mellon University. OPS5 had several advantages for the YES/MVS project:

1. It was flexible and modifiable.
2. It could be converted to run under LISP/VM [7] on an IBM computer.
3. Its use of production rules as a knowledge base representation seemed naturally suited to the type of knowledge occurring in the domain of computer operations.
4. Its *data-driven* form of inference was particularly appropriate to the situation of responding in real time to information received from the target MVS system.

These points are elaborated throughout the remainder of the paper. We now proceed to describe the basic OPS5 framework and reserve discussion of the extensions made to OPS5 to enable it to be used in a real-time application until Section 7.

An expert system written in OPS5 consists of three parts:

1. *Working memory*, in which all the data elements used or created during a problem-solving activity are stored.
2. *Knowledge base*, in which the expertise of the domain is encoded in the form of situation-action or IF-THEN production rules.
3. *Inference mechanism* or *"engine,"* which controls the sequence of activity of the expert system.

These three parts are maintained as separate entities. This separation represents an important distinction between such an expert system and a program written in a conventional procedural language, in which the knowledge and program control are intermixed and often difficult to distinguish.

```
(literalize printer-status          Name of working memory element class
          reliable?                 Is the information reliable?
          address                   Printer address
          forms                     Type of printer forms
          line-limit                Maximum number of lines allowed
          current-job               Job number of the current job
          status)                   Other status information
```

Figure 1

Definition of the "printer-status" class of working memory elements. Specific working memory elements of this class have values assigned to some of the six indicated attributes.

```
(p jm:printer-status-update                 Name of rule
  (task                                     Task identification
     ↑task-id jm:information-collection)
  (printer-status-reply                     Reply message from MVS
     ↑message-id iat8562                      Type of message is printer status
     ↑address <printer>                       Printer address
     ↑status <status>                         Printer status
     ↑forms <forms>                           Forms on the printer
     ↑current-job <job>                       Current job
     ↑line-limit <limit>)                     Maximum number of lines allowed
   (printer-status                          Printer status working memory
                                              element
     ↑address <printer>)                      Printer address
→
  (remove 2)                                Remove the reply
  (modify 3                                 Change the printer status working
                                              memory element
     ↑reliable? yes                           Mark the information as reliable
     ↑status <status>                         Update status
     ↑forms <forms>                           Update forms information
     ↑current-job <job>                       Update current job
     ↑line-limit <limit>))                    Update the line limit
```

Figure 2

Example of an OPS5 production rule from the JES queue space subdomain. This rule is used to update the status of information on a given printer when new information is received. Attributes are preceded by ↑ symbols. Attribute values may be bound to variables which can then be used in other patterns as constraints. In the case of values which are variable, the variable name is enclosed in < >.

Each working memory element is a member of a class. In each class definition an explicit list of attributes is given. Any particular working memory element belonging to that class has values attached to some of those attributes. **Figure 1** contains an OPS5 definition of the working memory element class "printer-status." Included in the definition is a list of six attributes which are associated with elements of that class.

The second component of an OPS5 expert system, the knowledge base, contains knowledge encoded in IF. . .THEN. . . production rules. The IF part (left-hand side, LHS) specifies the conditions which must be satisfied before the actions specified in the THEN part (right-hand side, RHS) can be taken. We refer to the members of the IF part as condition elements and of the THEN part as actions. **Figure 2** shows a production rule taken from the JES queue

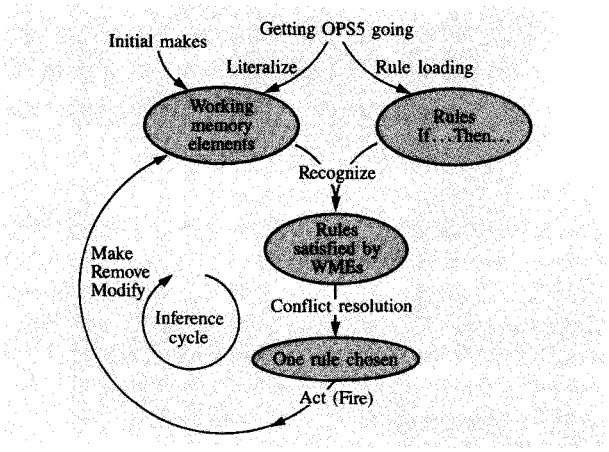

Figure 3

Basic OPS5 inference cycle.

space subdomain. In this rule, the IF part consists of three condition elements, one of class "task," 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 three working memory elements must be in working memory for the rule to be considered eligible by the inference engine. Furthermore, these three working memory elements must have mutual conditions that are satisfied. For example, both the "printer-status-reply" and "printer-status" working memory elements have an attribute "↑address" with value given by the same variable name <printer>. The value associated with this variable must be the same in both occurrences.

If the inference engine selects this particular rule for use, given that the condition elements have been satisfied, the actions specified on the right-hand side are carried out. Usually such actions call for the modification or removal of existing working memory elements (those referred to in the condition elements of the rule) or the addition of new working memory elements. The OPS5 terminology for these actions is MODIFY, REMOVE, and MAKE, respectively. In the case of this particular rule, two actions are present. The first calls for the removal of the "printer-status-reply" working memory element used to satisfy the second condition in the IF part. The second calls for the modification of the "printer-status" working memory element, by filling in new or modified values which are obtained from corresponding values in the "printer-status-reply" working memory element which matched the second condition element.

The OPS5 inference or control mechanism uses an inference cycle consisting of three phases:

1. Recognize,
2. Conflict resolution,
3. Act,

and is illustrated in **Figure 3**.

The IF parts (or left-hand sides) of all rules in the knowledge base are compared with the current contents of working memory to determine which rules have their condition elements satisfied (recognition phase). The Rete matching process [8] used in OPS5 is particularly efficient in carrying out this phase. A rule may have its IF part 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 such instantiations is called the *conflict set*. If the conflict set contains more than one instantiation, which is usually the case, then the control strategy is to select one of them to execute (conflict resolution). OPS5 provides two conflict resolution strategies: LEX (lexical), which is used in YES/MVS, and MEA (means-ends analysis). Both strategies prevent instantiations from being executed more than once, favor the most recently created data in working memory, and give preference to rules with more specific IF parts. The THEN part (or right-hand side) of the selected rule is executed (act phase). The actions of the THEN part normally change the contents of working memory (creating, modifying, or removing data elements). They may, however, also contain calls to LISP functions or to operating system facilities.

After the actions are carried out, the inference cycle is repeated. If no rules are found whose IF parts are satisfied, an OPS5 system halts.

5. YES/MVS organization

In order to be able to handle major incidents in the target system, YES/MVS runs in a separate computer and is not dependent upon the target system for computing time and other resources. Its sole interface to MVS is through an emulated JES3 console, appearing to MVS as a normal operator's console, but having the ability to be "read" and "typed on" by YES/MVS. YES/MVS runs in three concurrently running virtual machines under the VM/SP operating system [9]. Communication with the emulated JES3 console is provided by the Centralized Computer Operation Project (CCOP) facility [10]. (See **Figure 4**.)

1. The *Expert Virtual Machine* executes the rules in the knowledge base, receiving messages and submitting commands to the target machine. The expert system also sends text for, and receives responses from, the operator.
2. The *MVS Communications Control Facility (MCCF) Virtual Machine* provides the communications interface between the Expert Virtual Machine and the target system.

3. The *Display Control Virtual Machine* provides the communications interface between the human operator and the Expert Virtual Machine. It also transmits commands suggested by the expert system that are authorized for execution by the operator to MCCF. MCCF, in turn, sends these commands to the target system.

This design has the following benefits: better management of the different processes which are running asynchronously as parts of YES/MVS; relieving the expert system from low-level input/output considerations, such as message and display screen formatting; providing a self-contained knowledge base, so that operational policy description can be more easily read and modified. The three virtual machines are now described in more detail.

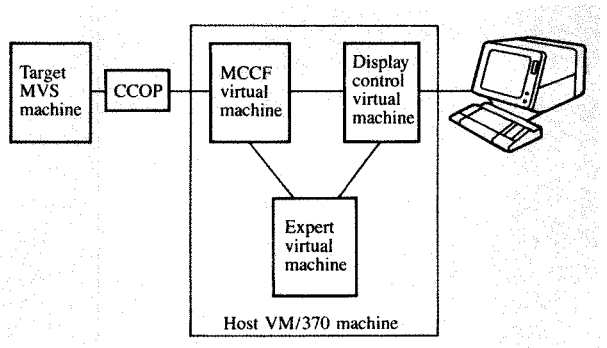

Figure 4

YES/MVS organization.

• *The Expert Virtual Machine*

The expertise in YES/MVS on system operation is encoded in an extended version of OPS5, which runs within the LISP/VM environment. Working memory contains all data used or manipulated in solving a problem, such as a model of all pertinent target system status data. All target system messages that are of importance to YES/MVS are automatically placed in working memory (after translation into the appropriate format by MCCF), where rules that respond to those messages are triggered.

In YES/MVS, the inference cycle never terminates. When there is no rule to be fired, the cycle remains in a wait state until an external message is received in working memory from the target system or from the operator or until the time arrives at which the creation of a new working memory element has been requested.

All the rules for different subdomains of the operator's actions coexist in one knowledge base. Some data describing the YES/MVS expert knowledge base are provided in **Table 1**. The average number of attributes per working memory class is 3.9. The maximum number of attributes in a working memory class is 21, and the minimum number is 1.

Three histograms in **Figure 5** give more detailed information about the matching contexts of all the rules in the knowledge base. Figure 5(a) displays a histogram of the rule set classified by number of condition elements. Figure 5(b) classifies the rule set by number of attributes. Figure 5(c) classifies the set of all condition elements by number of attributes.

Using the expertise from multiple subdomains together in one expert system has several advantages over creating a separate expert for each subdomain in its own virtual machine. Since rules in a given subdomain may use some of the same status information that is needed by rules in another subdomain, one global model of the target system is kept in working memory. This eliminates redundancy and, hence, inconsistency across subdomains in the expert system's model of the target system. This design additionally allows the expert system to control the scheduling of actions in different subdomains, rather than leaving such scheduling to the underlying VM operating system.

Table 1 Rule and working memory statistics for the expert virtual machine.

Subdomain name	*Number of rules*	*Number of WME classes*	*Number of attributes*
Batch scheduling	139	59	266
JES queue space	104	61	175
C-to-C links	68	42	150
Hardware errors	87	33	167
SMF management	25	29	103
Quiesce and IPL	52	51	185
Performance	41	16	66
Background monitor	32	23	105
Total	548	314	1217

• *The MVS Communications Control Facility Virtual Machine*

The second virtual machine runs the MVS Communications Control Facility (MCCF), which is written in REXX [11] and assembly language. MCCF controls the receipt of messages from MVS and the formatting of commands specified by the expert system. 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 Virtual Machine in OPS5 working memory format. This frees the knowledge engineer from concerns about parsing messages and extracting internal character strings.

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

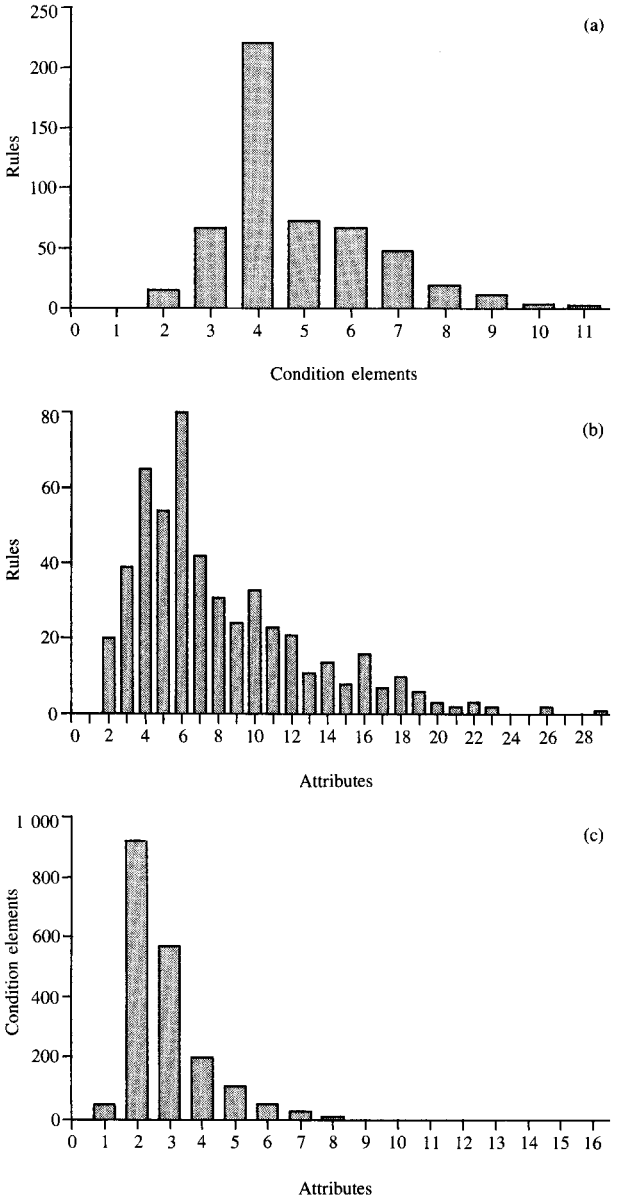

Figure 5

Matching contexts of the rules in the expert knowledge base: (a) Number of rules with a given number of condition elements; (b) number of rules with a given number of attributes; (c) number of condition elements with a given number of attributes.

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 Virtual Machine.

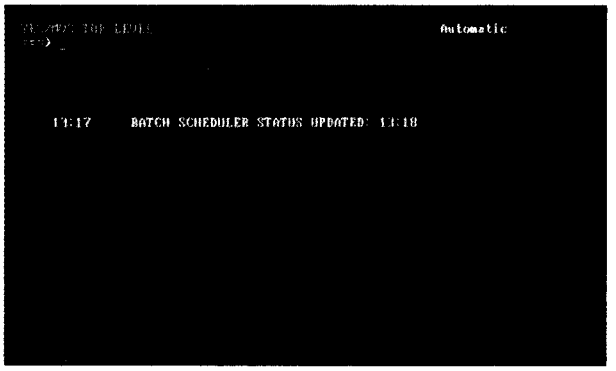

Figure 6

The YES/MVS top-level screen. The screen name is on the left side of the first line, the time of the day is in the center, and an indication of automatic mode and a pending count are on the right. The pending count indicates the number of messages which are not showing and which the operator has not yet acknowledged. Line 2 of the display provides a command line for typing (signaled by ===>) and is also used for the display of error messages. The bottom two lines of the display are used to provide information about the use of program function keys.

- *The Display Control Virtual Machine*

The third virtual machine controls the display facility and all interactions between YES/MVS and the system operator. The operator is provided with a hierarchical collection of screens. The screen at the top level (see **Figure 6**) provides an overview of incidents that do now or have recently required attention. The operator may select a problem for scrutiny, and a screen that is specific to that problem is displayed. In the case of an incident, such as depletion of JES queue space, the screen would contain a text description of the problem, a specification of the suggested response (including the specific commands to be submitted), a justification for the response, and a prognosis of the situation. In the case of scheduling batch jobs, the detail screen, as illustrated in **Figure 7**, shows the status of jobs in the batch job queue.

This design was motivated by discussions with system operators who indicated that, at a given time, they either were taking a broad overview of the target system or were concentrating on one particular situation.

YES/MVS can (as controlled by software switches that may be reset by the operator) either take actions automatically or give advice to the operator on how to handle situations. This is required because large computing centers are unlikely to turn system operation over to any software facility without a significant period of testing during which operators can maintain a manual override capability over the actions taken.

The actual implementation of the display facility was carried out through the use of an OPS5 rule base, combined with calls to the IBM Graphical Data Display Manager [12]

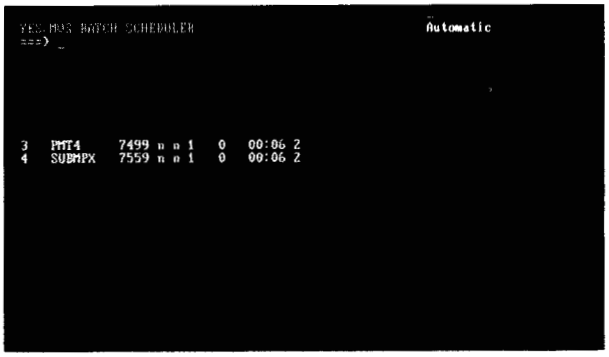

Figure 7

A lower-level YES/MVS screen. This screen gives detailed information about the status of the queue of large batch jobs. Each such screen can show up to 30 jobs arranged in two columns. For each job, the respective information shows its order in the queue, the job name, the job number, an indication of whether the user is waiting, an indication of whether the job should be scheduled even if it probably won't finish, how many jobs are ahead or at this point in the queue for this user, the number of days since the job was submitted, the estimated CPU time, and the job priority, if assigned.

for actual presentation on an IBM 3279 Color Display Station. The pertinent OPS5 statistics for the Display Control Virtual Machine are given in **Table 2**. The average number of attributes per working memory class for the display control knowledge base is 4.1. The maximum number of attributes in a working memory class is 23, and the minimum number is 1.

The three histograms in **Figure 8** are analogous to those given in Fig. 5 and describe the matching contexts of all the rules in the display control knowledge base. Color was used extensively in the design of the display screens at various levels. For example, in the screen showing the status of the batch job queue, jobs which are finished are shown in green, jobs which are running are shown in white, and jobs which are not yet run are shown in turquoise, in the order in which they will start (or have started) running. Jobs in hold status are displayed in yellow separately below the non-held jobs.

6. A detailed look at one subdomain

To give specificity to some of the issues raised in the preceding section, we look in more detail at the subdomain of scheduling very large batch jobs for overnight running.

• *Knowledge acquisition*

To develop the knowledge base for this subdomain, the operations staff and management were interviewed to determine what rules of thumb and procedures they used in doing the scheduling. The major factors that emerged were the following:

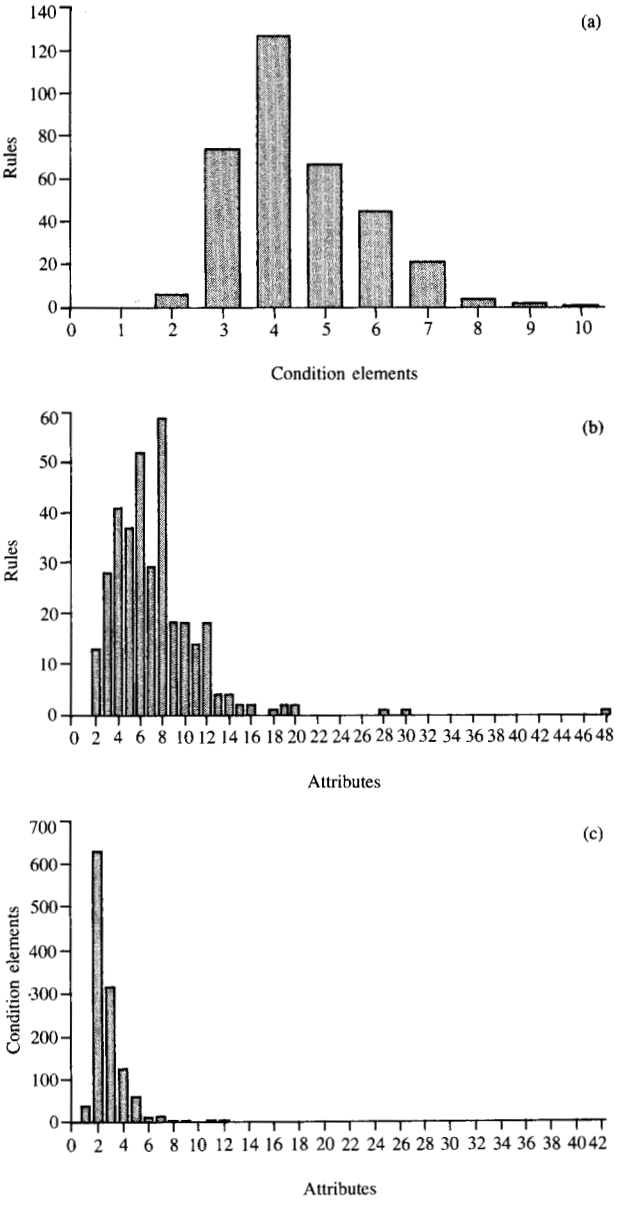

Figure 8

Matching contexts of the rules in the display control knowledge base: (a) Number of rules with a given number of condition elements; (b) number of rules with a given number of attributes; (c) number of condition elements with a given number of attributes.

Table 2 Rule and working memory statistics for the display control virtual machine.

Number of rules	*Number of WME classes*	*Number of attributes*
339	145	598

```
(p tms:order-nbr-wrong
  (task
    ↑task-id normal + 1)                    the task priority

  (job-order                                a job
    ↑next-job-id <next-job-id>                whose next job is <next-job-id>
    ↑in-q t                                   which is in the queue
    ↑order-nbr + 1 <order-nbr + 1>)           having an order-nbr + 1

{<nxt-job>
  (job-order                                the next job
    ↑job-id <next-job-id>
    ↑in-q t                                   in the queue
    ↑order-nbr <> <order-nbr + 1>)}           whose order-nbr isn't 1 more
                                              than the previous job's
→
  (modify <nxt-job>                         the strategy to rectify the
    ↑order-nbr + 1                            inconsistency: change the
      (compute 1 + <order-nbr + 1>)           order-nbr of the following job
    ↑order-nbr <order-nbr + 1>))              to be correct.
```

Figure 9

Truth maintenance rule – queue order numbers.

1. The overnight batch shift is brought up in the evening and brought down at the start of the day shift. This contrasts with many MVS installations which do not have explicit shift changeovers. This brought about requirements to
   a. Not schedule a job that would not be completed before the scheduled shift changeover.
   b. Run the larger jobs at the beginning (to give them enough time to be completed).
   c. Save the shorter jobs till near the end of the shift, to better fit the available time left in the shift.
2. Some users submit several (five, for example) multi-hour jobs; they expect that they will perhaps be run over several evenings (or if the shift load is light, they may be run in one night). The operations staff has a policy of fairness among users; they do not let five of user A's jobs be completed while running none of user B's jobs. This requirement was eventually codified into a round-robin scheduling strategy: Don't run job N for user A if user B hasn't had his job $N-1$ run yet (assuming, of course, that user B has that many jobs).
3. The jobs left in the system from previous days are given preference over subsequently submitted jobs.
4. Jobs having the same job name are run serially.
5. If a user has called the operator and asked for special treatment (perhaps he or she is working late), the operators often give his or her job(s) preference.
6. The shift start and stop times can change dynamically; for example, a hardware error may require the system to be given over to the repair personnel at 3 A.M., thus shortening the shift.
7. New jobs are dynamically scheduled in as they arrive.
8. Normally, four jobs are allowed to run at once; that is, the number of large batch initiators is four. However, this number is sometimes changed by the operator.

• *Knowledge base organization*

The rules comprising the knowledge base for the Batch Scheduler subdomain can be thought of as being divided into six groups:

1. Creating and maintaining a model of MVS and the jobs in the queue.
2. The scheduling rules.
3. The job priority setting rules.
4. Transmitting the results of the rules to the operator and/or MVS.
5. Shift state control (in shift, not in shift, starting, terminating).
6. Utility subroutine rules.

During operation the Batch Scheduler continually strives to derive a consistent model of what the batch job queue order should be, based on (changing) information from MVS. This approach is an implementation of a type of a Truth Maintenance System (TMS). (See [13] for further details.) In our case we have a model of a real-time system. As the system changes through time, the "facts" representative of the system change. Contradiction occurs between the changed facts and the previously derived consequences. Truth Maintenance is the removal of now-inconsistent deductions and the computation of new consequences in accordance with the changed facts, thus re-achieving a consistent state.

An example of a Truth Maintenance rule is provided in **Figure 9**. This is the case of maintaining the jobs to be run in an ordered queue. Each job in the queue has a pointer to the following job. Each job also has an order number, the sequence number of that job in the queue. When a new job is inserted in the queue (or a job is removed), the order numbers need to be adjusted. The rule in Fig. 9 keeps the order numbers "truthful." The left-hand side (LHS) of the rule is the recognizer of inconsistent data in working memory, and the right-hand side (RHS) provides a strategy for changing the state of working memory to achieve consistency (as defined by the LHS).

The rule recognizes an inconsistency when two adjacent jobs in the queue have order numbers that are not separated by 1. When a job is inserted, this rule fires repeatedly, advancing up the queue, until all the jobs have had their order numbers properly reestablished.

7. Real-time control

The MVS system being monitored and controlled by YES/MVS is, of course, highly dynamic. Problem states may be entered spontaneously; problems may disappear in the

middle of attempted resolution. In this sense, the MVS world is highly nonmonotonic. It is impossible to maintain an accurate model of MVS that is complete in all detail. Instead we maintain a model that provides a reasonably good description of the status of MVS, from the viewpoint of operations. The model is updated whenever MVS gives pertinent status information, either voluntarily, based upon responses to queries, or upon acknowledgment messages to control commands. Queries of status information are submitted at regular intervals or may be triggered by events and the need for information in the resulting analysis. The frequency of different queries varies enormously based on the volatility of the status data involved and on the requirement for current information. Extensive use of timestamps and validity flags provides additional information on the "currentness" of MVS status.

The status model of MVS is updated only on the receipt of information from MVS. Attempts to compute status from history and the anticipated response to stimuli are avoided, because of the many possibilities that exist for a stimulus not to have the intended effect. These include delays in command submission or processing, conflicting commands from operators, and nonresponse or errors in response to advice by operators. When YES/MVS is providing advice, as opposed to submitting control commands directly, there is a potential race condition between the existence of a problem state and the submission of a corrective command. It should be noted that this is an inherent problem, and the use of an automatic control system such as YES/MVS improves rather than exacerbates such situations.

The next sections identify specific requirements of an inference system which is to perform continuous, real-time, interactive control, and describe solutions in terms of various extensions to OPS5. Some of these extensions take the form of new primitives; others are LISP functions and macros added to the OPS5 environment.

- *Responsiveness*

The ability of an inference engine to process rules at a speed commensurate with real-time control is a basic concern. We have improved the speed of execution of OPS5 by compiling the right-hand side (RHS) of the rule. (Such a compilation process has been independently introduced in YAPS [14] and in OPS83 [15].) The matching process has been tuned with several LISP macros. Also, we distribute the rules among multiple OPS5 systems using concurrent processes in the form of separate virtual machines supported by a host computer.

- *Timed productions*

Being able to initiate an action at a given time is one of the fundamental requirements of a real-time control problem. With a data-driven inference engine, this includes the production of working memory elements at some future time. We accomplish this by defining a new RHS action primitive for delayed production, TIMED-MAKE, which takes the normal OPS5 MAKE arguments followed by either an absolute or a relative time specification. For example, execution of an RHS action,

```
(TIMED-MAKE query-request ↑type jes-q-space (in <iv>
  <un>))
```

would cause the production of a working memory element of class "query-request" with the value "jes-q-space" assigned to the attribute "type" after a certain elapsed time, which is specified by the values assigned to the variables <iv> and <un>, giving the interval and unit of time, respectively.

A timer function and timer queue were added as necessary support functions for the TIMED-MAKE action. To support debugging, functions were provided to manipulate the timer clock.

- *Communications*

Another requirement of real-time processing is the ability to have distributed processes interact in a timely fashion. Fast communication is achieved by introducing a new communication phase in the normal OPS5 inference cycle (recognize, conflict resolution, act). During the communication phase, external messages are picked up and outbound messages are sent. Conflict resolution then takes place based on changes to working memory as the result of both RHS actions and incoming messages.

All messages are sent out by a communication primitive, REMOTE-MAKE, which takes the same arguments as the regular OPS5 MAKE action, with an additional attribute "↑Rm-to:" whose value is the user-id of the intended receiver virtual machine. The message is actually sent by the host system's program-level message sending mechanism. The "↑Rm-to:" attribute-value pair is changed, en route, to another attribute "↑Rm-from:" with the sender's machine user-id as its value. In the following example, a REMOTE-MAKE action is being sent to the virtual machine specified by the value of the variable <query>. In the receiver virtual machine, a working memory element of class "tape-drive-status-query" is to be created with attribute "↑address" and value specified by the variable <tape>.

```
(REMOTE-MAKE tape-drive-status-query ↑rm-to: <query>
  ↑address <tape>)
```

The REMOTE-MAKE action, as is the case with the TIMED-MAKE action, can use any of the OPS5 functions to create result elements. Thus, one can write a meta-level REMOTE-MAKE rule, if desired, to create messages dynamically from templates, defaults, and substituted values of bound variables.

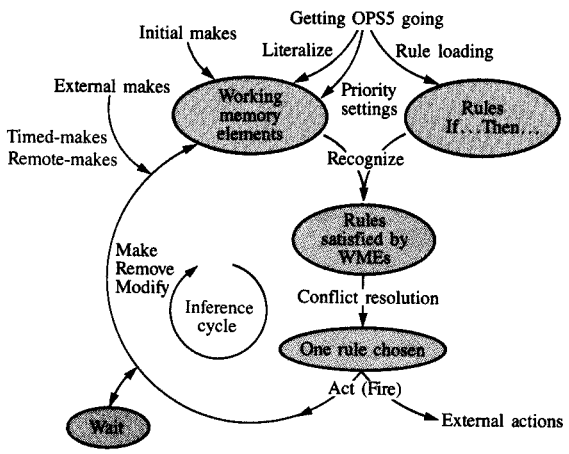

Figure 10

YES/MVS inference cycle.

For debugging purposes, a global variable can be set to block the actual transmission of elements created by REMOTE-MAKE. Then the messages are displayed along with requests for replies. When a reply is entered or selected from a pre-existing file using a multi-window interactive editor, it is employed just as if it came from another virtual machine.

• *Need for explicit control*

There are critical problems that require a command sequence to be issued to MVS without other queries or commands being interspersed. Hardware error message handling is one such case. Such a real-time requirement necessitates explicit control over the rule firing in the inference engine. For this purpose, the two modes of OPS5 conflict resolution, LEX and MEA, were extended by a Priority Mode which has precedence over these.

To implement the priority mechanism, each rule has an additional left-hand side (LHS) condition element (TASK ↑task-id XXX), where XXX is a unique task name or a list (expressed as an OPS5 disjunction) of task names to which the rule is relevant. Each such task-id XXX has an associated priority. The conflict resolution phase of OPS5 is modified so that the active conflict set is temporarily reduced by excluding all active rules that do not have the highest-priority task among the set. Then, the normal OPS5 conflict resolution process acts on this reduced set. The task working memory elements as well as associated priorities are defined either by a top-level MAKE or by an RHS action. Tasks can thus be dynamically created or destroyed. The priority can also be dynamically computed as an RHS action of a rule.

The modified inference cycle used in YES/MVS is shown in **Figure 10**, indicating the introduction of external and timed events and the use of priorities. As an alternative to implementing the priority control mechanism, the MEA strategy could have been used for this purpose by carefully making ordered "goal" working memory elements and using them for priority control, having them be the first condition element of the rules. However, this would usurp the MEA mechanism for this purpose and make it unavailable for other uses. Also, the priority of a rule would be obscure, being dependent on the order in which the MAKE actions of the MEA goals were done.

The priority control mechanism effectively satisfies our real-time control needs. It also allows control over rule interaction between different subdomain areas. While meta-rules could have been used to refine the conflict resolution strategy, we found that the priority control mechanism offered an equivalent capability without incurring the overhead and complexity of such a facility. (Benjamin and Harrison [16] use meta-rules for a different purpose: reasoning about the contents of the conflict set.) Furthermore, it provides a rule-grouping paradigm similar to the use of contexts in EMYCIN [17] and rule-groups for HH rules in EXPERT [18].

• *Requirements for continuous operation*

There are at least three basic requirements for operating in a continuous mode:

1. The inference engine should not terminate when no rule is eligible to fire. We implemented an OPS5 rule OPS-WAIT, which puts the system into a waiting mode. Any external message (including a timer event) causes the system to resume, with the new data added to working memory.
2. The system should ideally run on a special-purpose, high-availability computer, different from the subject machine. If the host computer itself or the virtual machines comprising the system go down, the system must be restarted. We call an automatic restart procedure during the host computer initial program load and also when a down machine is detected during a periodic mutual polling among virtual machines of the system.
3. Working memory elements that have served their purposes must be removed. The accumulation of old useless data in the working memory not only creates a memory space problem in continuous operation, but, of more importance, instantiates the wrong productions in a data-driven inference engine, such as OPS5. We have made use of many different "garbage collection" techniques (RHS actions) to remove old data, including the one illustrated next.

Removal of multiple working memory elements must be done carefully so as not to trigger rules unintentionally which might be satisfied when only a partial set of working

memory elements had been removed. For example, the ability of a rule to fire may depend not only on the presence of some elements, but also on the absence of others. The priority mechanism can be used to cause an atomic procedure, as shown in the three-rule example in **Figure 11**. (This also illustrates the dynamic creation of tasks.) Suppose the normal operating priority is 100. The priority for the CLEAN-UP task would be set low, say, at 50. Define another task name, IN-CLEAN-UP, with a priority, say, 150, which is higher than the priority of other tasks. The CLEAN-UP task is created in the system as a permanent working memory element during initialization.

The final rule in Fig. 11 is less specific than the rule above it and so does not fire until all garbage has been removed, due to the conflict resolution mechanism of OPS5.

```
(p start-clean-up                          Low-priority rule
   (task ↑task-id CLEAN-UP)                that fires when no
   →                                       other normal action
   (make task ↑task-id IN-CLEAN-UP)        rules fire.

(p doing-clean-up                          This rule repeatedly
   (task ↑task-id IN-CLEAN-UP)             fires and removes
   {<garbage> [List of working memory      all garbage as an atomic
       element names to be removed]}       procedure, at high
   →                                       priority.
   (remove <garbage>))

(p clean-up-done                           This rule removes the
   {<done-task>                            IN-CLEAN-UP task which
   (task ↑task-id IN-CLEAN-UP)}            is now garbage and the
   →                                       system reverts back to a
   (remove <done-task>))                   low-priority CLEAN-UP mode.
```

Figure 11

Three rules illustrating the collection of unneeded working memory elements as an atomic action.

8. Developing YES/MVS

From the knowledge engineer's perspective, it is more difficult to develop rules that interact with both the target system and the operator than to develop rules that assume full control over the target system. Interaction with the operator requires that all actions not only make sense to the target system, but also are reasonable in timing and quantity to the operator.

- *Knowledge acquisition*

Domain knowledge sources for YES/MVS included one operations expert dedicated full-time to the project, MVS operators and operations managers, resident systems programmers, system manuals, operator console traces of responses to actual problems, and, occasionally, the designers of the MVS operating system itself.

Intensive meetings with the primary expert culminated in the formulation of strategies for problem handling in each of the selected subdomains. Knowledge engineers then produced documents detailing these strategies and distributed the documents to several experts for feedback. This served the dual purpose of correcting misconceptions and of uncovering considerations that had not yet surfaced. Observation of operator actions and the examination of console traces by knowledge engineers unearthed additional details that the experts had omitted or not emphasized. A common problem in expert systems development is that experts often fail to describe knowledge completely, at least in the initial phases of a project. In part, this is what motivates a software organization that facilitates incremental development.

Rules were coded in each of the subdomains. The rule coding process was facilitated by a programming environment in which the LISP/VM system, on which OPS5 was built, and the system editor exist as co-routines. Thereafter followed a period of iterative information gathering, testing, and debugging. Information gathering took the form of additional discussions and "rule walkthroughs" with the operations staff. Systems programmers were consulted in cases of inadequate information or conflicting viewpoints.

- *Testing*

It was established early in the project that the complexity and dynamics of the MVS environment would prohibit the construction of an MVS simulator against which to test YES/MVS. Testing on a system that is not in production use frequently omits much of the complexity of a running production environment. Thus, the primary testbed for YES/MVS has been the actual operations environment of a production system. Still, there have been challenges. It is hard to manufacture some kinds of target system problems, for example, hardware failures.

Balancing testing objectives with service degradation presented another testing consideration. While tests involving "sabotage" of the target system were performed off shift, some users were nonetheless affected. Also, the dynamics of a production workload make problem re-creation difficult when tests of modifications to the knowledge base are being made.

9. Operation and evaluation of YES/MVS

During initialization, the central processor utilization averages 0.5% of an IBM 3081-K processor for the Expert Virtual Machine and 0.86% for the Display Control Virtual Machine. During normal running, the average usage is 0.3% and 0.57%, respectively.

YES/MVS ran regularly at the Yorktown Computing Center during a period of over nine months. (The functions and capabilities of YES/MVS are currently being replaced by a second version.) YES/MVS itself enjoyed reasonably high availability, although this result was achieved only after considerable effort. For example, rules were added to the

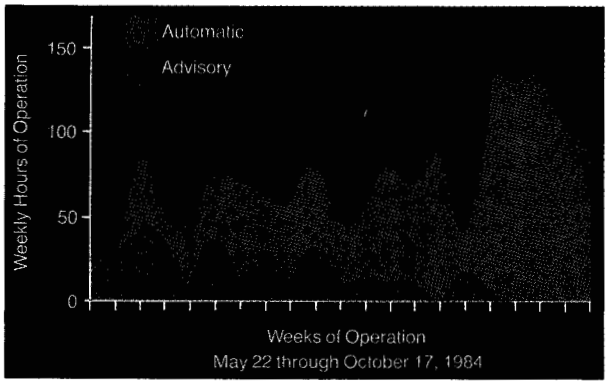

Figure 12

YES/MVS operations statistics.

background monitor subdomain to test the status of the other two virtual machines. Checkpoint and recovery procedures were added as well. Tests were built into the Expert Virtual Machine to ensure that errant messages were accounted for and retries sent, if necessary. Generally, YES/MVS went down only when the host computer went down.

Eventually, YES/MVS ran fully authorized, taking action automatically and subsequently notifying the operator. Operations staff members were enthusiastic about the project and informally reported that YES/MVS successfully detected and responded to problems in the subdomains implemented. For example, YES/MVS had alerted operators to channel-to-channel link problems of which they were unaware and guided the operators in resolving these problems. In developing YES/MVS the best expertise was sought, and operators have reported learning better techniques for handling problems by following the advice provided by YES/MVS. The YES/MVS Batch Scheduler and SMF Manager were routinely used and were significant productivity aids to operators.

10. Project history

The project was begun early in 1983 as a collaborative effort on the part of the Expert Systems Group and the Expert Systems for Systems Management Group. The group began as a six-person effort, augmented by a seventh person from the computing center who joined the group on sabbatical assignment and served as the resident expert. By late 1983 two additional members had been added.

A three-day work session at the beginning of the project determined the scope of the initial effort in terms of the subdomains to be covered. Responsibility for each subdomain was divided among the project members. Demonstrations of two of the domains, JES queue space monitoring and batch job scheduling, together with the operator's console display, were first given in the fall of 1983, using a simulated set of messages from MVS. Testing during the fall was primarily limited to exercising individual subdomains with canned data. In February 1984, on-line testing of an integrated system was begun, using a schedule of two evening sessions each week. On April 12, electronic mail was received from a member of the operations staff commenting positively about the advice provided by YES/MVS concerning the canceling and restarting of channel-to-channel links. By May, the system was providing advice to operators, using the knowledge bases for some subdomains, while testing continued in others. Statistics on YES/MVS availability began to be gathered. (See **Figure 12**.) The YES/MVS system was demonstrated during an IBM exhibit at the 1984 National Conference on Artificial Intelligence, held August 6–10 in Austin, Texas, emphasizing the two subdomains of JES queue space monitoring and batch job scheduling.

11. Conclusions

There is significant economic value in a facility that will increase either the productivity of computer operators or the quality of the function they provide. Still, there are inherent problems of

1. Complexity of the problem domain,
2. Variation in operational policy from one computer installation to the next, and
3. Evolution of the computer system and of operational policy at any one site.

In the development of YES/MVS, we have found the "IF condition, THEN take action" format of production rules to have advantages in addressing all of the three mentioned difficulties that are inherent in the operator's task.

The modularity of production rules encourages writing software that is relatively easy to read and modify. This should tend to simplify the long-term maintenance of an operator's knowledge encoded in production rules. It is also well understood that the process of encoding an expert's knowledge must be done incrementally, and the modularity of production rule software helps support this incremental development.

We found that our computer operators most frequently describe their knowledge in sentences such as "When I see this . . . and this . . . , then I do this . . . and this . . . ." Similarly, a standard statement of operational policy would be of the form "under circumstances . . . , take actions . . . in order to change from resource allocation scheme A to resource allocation scheme B." We conclude that production rules invoked through a data-driven inference mechanism provide a natural representation for these forms of operational knowledge.

Because production rules are a natural vehicle for stating operational policy and because YES/MVS has established

the feasibility of encoding the complexity of operators' knowledge about computing system problem diagnosis and recovery, additional work on YES/MVS has received strong support. A second version of YES/MVS is being developed by the Expert Systems for Systems Management Group. This second version will be installed at several large IBM computing centers. It will contain software to address problems that are sufficiently uniform that generally applicable knowledge can be used to control diagnosis and corrective actions. It will also emphasize software constructs that support the natural statement and direct implementation of knowledge about the operational policy that is unique to a particular computing installation.

Acknowledgments

The authors wish to acknowledge the substantial contribution of Barry Trager, for his efforts in converting OPS5 written in MACLISP to run under the LISP/VM system on VM/370. We would like to thank the computing center staff at the Thomas J. Watson Research Center for their participation in this project and for their patience during testing. We gratefully acknowledge the participation of the summer personnel who worked on the project. Finally, we thank all the people who provided expertise on the operation of the Yorktown JES3 system.

References

1. *OS/VS2 MVS/System Product—JES3 General Information Manual*, Order No. GC28-1025-11, December 1984, available through IBM branch offices.
2. William R. Nelson, "REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents," *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA, August 18–20, 1982, pp. 296–301.
3. Lawrence M. Fagan, "VM: Representing Time-Dependent Relations in a Medical Setting," Ph.D. Thesis, Stanford University, CA, June 1980.
4. Robert B. Wesson, "Planning in the World of the Air Traffic Controller," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, August 22–25, 1977, pp. 473–479.
5. C. L. Forgy, *OPS5 User's Manual*, Order No. CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, July 1981.
6. L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley Publishing Co., Reading, MA, 1985.
7. *LISP/VM User's Guide*, Order No. SH20-6477-0, July 1984, available through IBM branch offices.
8. C. L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artif. Intell.* **19**, No. 1, 17–37 (September 1982).
9. *VM/SP Introduction*, Order No. GC19-6200-2, November 1984, available through IBM branch offices.
10. A. A. Guido, "Unattended Automated DP Center Operation: Is It Achievable?", *European GUIDE Proceedings*, Lyon, France, June 7–10, 1983, pp. 440–446.
11. *VM/SP System Product Interpreter Reference (Release 3)*, Order No. SC24-5239, September 1983, available through IBM branch offices.
12. *Graphical Data Display Manager Programming Reference*, Order No. SC33-0101-4, November 1984, available through IBM branch offices.
13. M. Schor, "Using Declarative Knowledge Representation Techniques: Implementing Truth Maintenance in OPS5," *Proceedings of the First Conference on Artificial Intelligence Applications*, Denver, CO, December 1984, pp. 261–266.
14. Elizabeth Allen, "YAPS: A Production Rule System Meets Objects," *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC, August 22–26, 1983, pp. 5–7.
15. C. L. Forgy, *The OPS83 User's Manual*, Production Systems Technology, Inc., Pittsburgh, PA, 1985.
16. D. P. Benjamin and Malcolm C. Harrison, "A Production System for Learning Plans from an Expert," *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC, August 22–26, 1983, pp. 22–26.
17. William J. van Melle, *System Aids in Constructing Consultation Programs*, UMI Research Press, Ann Arbor, MI, 1981.
18. S. M. Weiss and C. A. Kulikowski, "EXPERT: A System for Developing Consultation Models," *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan, August 20–24, 1979, pp. 942–947.

Received April 12, 1985; revised July 15, 1985

Robert L. Ennis *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Ennis joined IBM in 1966 as a computer operator. In 1970, he was appointed operations manager for the second and third shifts and in 1973 he became first shift manager for the Research Division Computing Center. He was manager of the operations support group from 1975 to 1983. Currently he is working in the expert systems area of the Computing Science Department.

James H. Griesmer *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Griesmer is a Research staff member in the Computing Technology Department. He has been a member of the Research Division since 1957 and has carried on research activity in the areas of game theory, switching theory, circuit fault diagnosis, error-correcting codes, computer algebra, and, most recently, expert systems. He was a member of the technical staff of the Director of Research, manager of the Research Computing Center, manager of a project which developed an interactive symbolic mathematical system called SCRATCHPAD, and manager of education and development programs for the Yorktown laboratory. In 1970–71 he was a Visiting Mackay Lecturer at the University of California at Berkeley in the Department of Electrical Engineering and Computer Science. Dr. Griesmer received a B.S. in mathematics from the University of Notre Dame, Indiana, in 1951, and a Ph.D. in mathematics from Princeton University, New Jersey, in 1958. He is a member of the Association for Computing Machinery and the American Association for Artificial Intelligence. Dr. Griesmer served as Chairman of the Special Interest Group on Symbolic and Algebraic Manipulation (SIGSAM) of the Association for Computing Machinery from 1973 through 1975.

Se June Hong *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Hong received his B.S. in electrical engineering from Seoul National University, Korea, in 1965, and his M.S. and Ph.D. in electrical engineering from the University of Illinois, Urbana, in 1967 and 1969. He then joined IBM at the Poughkeepsie, New York, laboratory, working in the areas of fault-tolerant computing and design automation. Dr. Hong joined the IBM Thomas J. Watson Research Center at Yorktown Heights in 1978. He is currently a senior manager in the Computing Technology Department, responsible for projects in computer algebra, natural language, education, and knowledge-based programming. During the academic year 1974/1975, he was a visiting associate professor at the University of Illinois, Urbana. He

was a visiting professor at the Korea Advanced Institute of Science and Technology (KAIST), for the month of October 1980. Dr. Hong is a Fellow of the Institute of Electrical and Electronics Engineers and a member of the American Association for Artificial Intelligence, the Association for Computational Linguistics, the Association for Computing Machinery, Korean Engineers and Scientists in America, the Mathematical Association of America, and Sigma Xi. He received an honorable mention award for Outstanding Young Electrical Engineer in 1975 and three Outstanding Innovation Awards from IBM. He has served as a distinguished visitor in the Computer Society (1972–74), as the guest editor of the Reliable and Fault-Tolerant Special Issue of *IEEE Transactions on Computers* (July 1982), and a member of the Ad-hoc Visiting Team for the IEEE Engineering Accreditation Board (1979-84). Dr. Hong is currently a member of the Edison Medal Awards Committee.

Maurice Karnaugh *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Karnaugh has been a Research staff member at the IBM Thomas J. Watson Research Center since 1970. He is currently working in the field of knowledge-based systems. He received the Ph.D. degree in physics from Yale University, New Haven, Connecticut, in 1952. From 1952 to 1966, he worked at the Bell Telephone Laboratories, studying the application of digital techniques to telecommunications switching systems. From 1966 to 1970, he managed a number of telecommunications-related projects at the IBM Federal Systems Division. Dr. Karnaugh is an Adjunct Professor of Computer Science at the Polytechnic Institute of New York. He was elected a Fellow of the Institute of Electrical and Electronics Engineers in 1975 for contributions to the understanding and application of digital techniques in telecommunications. Dr. Karnaugh is a Vice President of the International Council for Computer Communication. He is a member of the IEEE Computer and Communications societies, the Association for Computing Machinery, the Special Interest Group on Artificial Intelligence (SIGART), Phi Beta Kappa, and Sigma Xi.

John K. Kastner *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Kastner is currently a member of the Research staff at the IBM Thomas J. Watson Research Laboratory. He received the B.A. in physics from Occidental College, Los Angeles, California, and the B.S. in engineering from the California Institute of Technology, Pasadena, both in 1974; the M.S. in computer science from the University of California at Los Angeles in 1977; and the Ph.D. in computer science from Rutgers University, New Brunswick, New Jersey, in 1983. His current research interests include artificial intelligence, medical modeling and decision making, and expert systems design.

David A. Klein *University of Pennsylvania, Philadelphia, Pennsylvania 19104.* Mr. Klein is currently 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.

Keith R. Milliken *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Milliken is a Research staff member at IBM's Thomas J. Watson Research Center. He is manager of the Expert Systems for System Management Project, involved with the development of experimental expert systems to assist in the management of large computing systems. Dr. Milliken received a bachelor's degree from Occidental College, Los Angeles, California, 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 at Princeton, New Jersey. From 1975 to 1979 he was an Assistant Professor of Mathematics at California State Polytechnic University, Pomona. He joined IBM in the spring of 1979. His interests include expert systems, the management of large computing systems, and combinatorial mathematics and algorithms. He is the author of research papers in each of these areas.

Marshall I. Schor *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Schor is a senior programmer in the Research Division, managing a group working on expert systems development and delivery environments. He joined IBM in 1968 as a junior engineer and spent twelve years in disk drive attachment development. He was the group leader and subsequently the manager for the prototype project that led to the data-caching models of the IBM 3880 control units. Following a two-year assignment with the Corporate engineering, programming, and technology staff, he joined the Research Division in Yorktown, working on expert systems. Mr. Schor received a B.S. in engineering from the California Institute of Technology, Pasadena. He is a member of the Association for Computing Machinery, the Institute of Electrical and Electronics Engineers, and the Special Interest Group on Artificial Intelligence (SIGART).

Hugo M. Van Woerkom *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Mr. Van Woerkom is an advisory programmer in the Systems Analysis Department. He has been a member of the Research Division since 1983. Before joining the Thomas J. Watson Research Center, he worked in systems and scientific programming in Kingston, New York. Mr. Van Woerkom received a B.A. in philosophy from Carroll College, Helena, Montana, in 1961, an M.Div. from St. Thomas Seminary, Denver, Colorado, in 1965, and an M.S. in computer science from Syracuse University, New York, in 1984.