YES/MVS: A Continuous Real Time Expert System

J.H. Griesmer, S.J. Hong, M. Karnaugh, J.K. Kastner, M.I. Schor
Expert Systems Group, Mathematical Science Department
R.L. Ennis, D.A. Klein, K.R. Milliken, H.M. VanWoerkom
Installation Management Group, Computer Science Department
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

ABSTRACT:

YES/MVS (Yorktown Expert System for MVS operators) is a continuous, real time expert system that exerts interactive control over an operating system as an aid to computer operators. 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.

I INTRODUCTION

Expert systems techniques are beginning to be successfully applied to real problems in industry, although only a handful are reportedly in use so far. Most of the applications are consultation oriented, run in a session or in a batch mode, and deal with a static world. The nuclear reactor monitoring expert system, REACTOR [1], and the patient monitoring expert system for intensive care units, VM [2], are among the few attempts at continuous on-line operation and real time processing. However, neither of these systems exercise any real time interactive control over the subject being monitored. The Yorktown Expert System for MVS operators (YES/MVS) is a real time interactive control system that operates continuously.

The idea of on-line monitoring or controlling of one computer by another is not new. Watch-dog processors [3] and maintenance processors [4] have been designed to assist in the recovery from software errors and hardware errors while the subject computer is in operation. What is new is the application of an expert system approach to the control of computer operations.

A. Importance of the Domain

Computer operations is a monitoring and problem solving activity that must be conducted in real time. It is becoming increasingly complex as data processing installations grow. Large data processing installations often involve multiple CPU's and a large number of peripherals networked together, representing a multi-million dollar investment. Many of the installations run real time applications (e.g., banking, reservations systems). The control of a typical large system rests largely in the hands of just a few operators. Besides carrying on such routine activities as mounting tapes, loading and changing forms in printers, and answering phones, an operator continuously monitors the condition of the subject operating system and initiates queries and/or commands to diagnose and solve problems as they arise. A long training period is required to produce a skilled operator; trained operators, in turn, are often promoted to systems programmers. The resulting shortage of skilled operators and the increasing complexity of the operator's job calls for more powerful installation management tools. We have chosen the management of a Multiple Virtual Storage (MVS) operating system, the most widely used operating system on large IBM mainframe computers, as an example of the application of expert systems to problems in computer installation management.

B. Use of Expert System Techniques

Each installation has a different configuration and different local policies for computer operations, both of which change over time. The software running in a large computer installation represents hundreds of man-years of development and is comprised of many interacting subsystems. To deal with such complexity, operators and system programmers often rely on many "rules of thumb" gained through experience. The development of installation management tools which can be easily tailored and modified, and which can incorporate such "rules of thumb" are highly desirable. An expert systems approach was a natural choice because of its flexibility and maintainability.

C. Special Challenges

There are many new requirements in building a real time expert system to assist a computer operator. The environment is too complex and dynamic to allow for obtaining information by querying the human operator. Unlike many of the consultation style expert systems (e.g., MYCIN [5], CASNET [6]), this means that conclusions are based on primitive facts obtained directly from the system being monitored and not from human interpreted inputs. In addition, the dynamic nature of the subject system introduces potential inconsistencies in the expert system's model of the subject MVS system. By the time conclusions are to be put into effect (recommendations made, actions taken), many of the facts from which those conclusions were derived may have changed. This complexity and dynamic character make it very difficult to simulate or model the subject system. Developing and debugging such an expert system presents yet another interesting challenge.

D. The OPS5 Base

To be able to handle real time on-line data, the inference engine needs to be mainly data driven (also recognized by REACTOR [1] and VM [2]). The OPS5 production system developed by C. L. Forgy [7] was chosen as our tool primarily for this reason. Also, significant applications based

on the OPS family of production systems have been reported (R1/XSEL/PTRANS [8-10] and ACE [11]). Of importance was our ability to convert OPS5 to run in our computing environment. While OPS5 was not directly usable for our application, it possessed the important properties of flexibility (stemming from being a low level language) and easy modifiability. We have made a number of important extensions to OPS5 so that our continuous real time requirements could be met.

II THE DOMAIN

The MVS operating system running with a Job Entry System (JES), puts out various system messages to the operator. While there are literally hundreds of different types of messages, the number that are relevant to an operator is much smaller. The majority of purely informational messages may usually be statically filtered and diverted to a log. Even then, the peak message rate from MVS to the operator sometimes exceeds 100 a minute (e.g., a response to an operator request for a list of jobs in a particular category).

When a (potential) problem is detected, the operator may query MVS for additional information and send one or more corrective commands. The operator must often anticipate informational needs and dynamically keep track of a number of relevant status variables (out of thousands).

There are many different subdomains in the domain of operator activities. The six subdomains described below were selected for early implementation, since they touch a majority of those operator activities which involve no physical intervention.

A. JES Queue Space Management

All batch jobs processed under MVS are staged from a central spool file, called the Job Entry System (JES) queue space, before, during, and after execution. The operator is concerned with the remaining available queue space, because the job staging subsystem, JES, cannot recover if queue space is exhausted. When the level of remaining queue space becomes critically low, many actions are initiated to free additional space, such as forcing the printing of jobs which have finished execution and dumping large print jobs to tape. In extreme cases, the system can 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 subject 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.

B. 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 contributes 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.

C. 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. These 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 require some other special consideration, running longer jobs early in the shift and running only those jobs that can finish 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. A separate paper [12] describes a truth maintenance approach using OPS5 for keeping a dynamically correct priority assignment of the jobs.

D. 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), 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 the OPS5 working memory. The hardware configuration data is initially loaded from the files used in the MVS system generation process.

E. Monitoring Software Subsystems

The main activity in this area is to generate informative incident reports for the systems programmers who are responsible for specific software subsystems. When an incident occurs, such as an abnormal end of execution, relevant information is captured and an appropriate incident report is prepared. In limited cases, reallocation of resources may allow recovery from software failures.

F. 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. An eventual goal is to diagnose the cause of performance problems and to take corrective actions.

III THE YES/MVS SYSTEM

A. System Organization

YES/MVS runs under the VM/370 operating system on an IBM 3081 computer. Because YES/MVS and the subject MVS system are resident in different computers, problems in MVS do not interfere with the operation of YES/MVS, and MVS can continue to operate under manual control should the YES/MVS host experience difficulties.

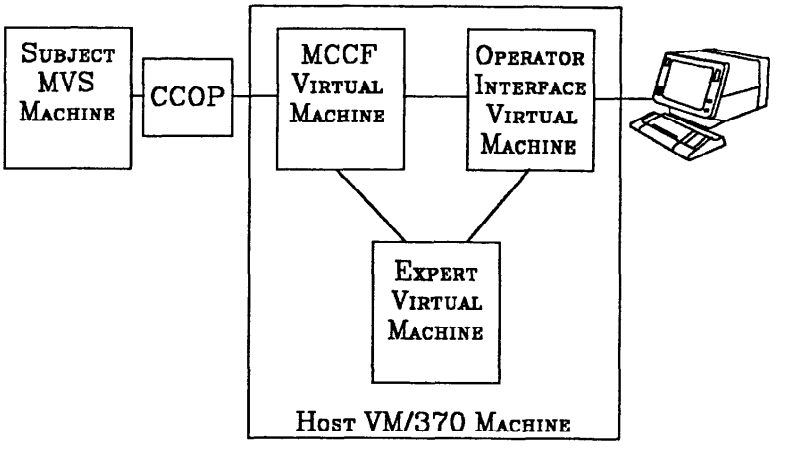

Figure 1. YES/MVS System

YES/MVS is presently partitioned into three virtual machines for speed as well as functional separation. One of these contains the MVS operator expert, the second one contains the MVS Communications Control Facility (MCCF), and the third one is used to control the YES/MVS operator's display console. MCCF communicates with the MVS system through a separately developed facility, called CCOP [13]. CCOP provides centralized control and filtering of messages between the various computers in an installation and their operators.

Intelligence in the form of OPS5 rules is distributed between the expert virtual machine and the display controller. These machines communicate with each other via the REMOTE-MAKE mechanism, which is described in the next section. MCCF is implemented in several thousand lines of the system exec language, REXX [14].

MCCF acts as a message filter and also translates the messages between the expert system and MVS. For example, a JES command generated by the MVS operator expert,

```
(reroute-print-job-from-3211-to-3800-printer)
```

together with three parameters: job name, 3211 address, and 3800 address, is translated and sent to JES as

```
8f u j=SAMPLE, d=6C0, nd=00E
```

B. Operator's Console

The YES/MVS operator console displays one-line messages on the top level input screen describing events relating to the various tasks YES/MVS is concerned with. The operator may select one of the displayed messages and request further detail. The detail level screen contains the recommended action or information along with an explanation. If an action is called for, the operator is given the choice of automatically issuing the command (U-DO), showing that he or she will manually type the recommended command at another terminal (I-DID), or rejecting the command being proposed (NO-DO). If the command is rejected (NO-DO), the operator is prompted to enter the reasons for the rejection which is fed back to the YES/MVS knowledge engineers. The action screen is intended only for use during a pre-certification phase. Once a particular command is certified, the operator display machine will send the commands to MCCF without asking the operator. For certified commands, the detail screen only displays information on the action taken along with a justification. Another second level screen allows operators to enter unsolicited information or requests.

```
   YES/MVS TOP LEVEL          16:14                  Pending:   0
===>
15:57    BATCH SCHEDULER STATUS UPDATED:  16:09
15:57    SMF: CHECK STATUS OF SMF DATASETS
16:09    BATCH SCHED: MODIFY JOB-ID 5003 TO PRIORITY 14
16:10    SMF: ENTER DUMP FOR SYS1.MANA
16:11    CTCFIX: RESTART COMMUNICATION TO YKTVMZ




PF01 PF02 PF03 PF04  PF05  PF06 PF07 PF08 PF09 PF10  PF11  PF12
     U.I. EXIT WRKNG SELCT DONE BACK FWD       RFRSH ERRST HOME
```

Figure 2. YES/MVS Operator Console Top Level Screen

IV CONTINUOUS, REAL TIME, INTERACTIVE CONTROL ISSUES

The MVS system being monitored and controlled by YES/MVS is a dynamic world. Problem states may be entered spontaneously. Also, a problem may disappear in the middle of the solution process. In this sense, the MVS world is highly non-monotonic. 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 volunteers pertinent status information, based upon responses to queries and upon acknowledgement 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. This is because of the many pitfalls that exist for a stimulus not to have the anticipated effect. These include delays in command submission or processing, conflicting commands from operators, and non-response or errors in response by operators to advice. It is especially the case that, 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.

We now 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.

A. Speed Considerations

The ability of an inference engine to process in real time is a basic concern. We have improved the speed of execution of OPS5 by compiling the right hand side (RHS) or consequent part of a rule. (Such a compilation process has been independently introduced in YAPS [15], and in OPS83 [16].) The matching process has been tuned with several LISP macros. The modified version of OPS5 runs significantly faster than other LISP implementations of OPS5. Also, we distribute the rules among multiple OPS5 systems using concurrent processes in the form of separate virtual machines supported by a host computer.

B. 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 (WMEs) 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 a time specification. (The OPS5 MAKE action creates new elements and adds them to working memory.) For example, execution of an RHS action,

```
(TIMED-MAKE  AAAI  ↑due-date  past
                (AT  TIME: 1700  DATE: 84 4 2))
```

would cause the production of a WME, named AAAI, at 5 p.m. on April 2nd, 1984, with the value "past" assigned to the attribute "due-date".

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 and pop the timer queue as needed.

C. 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.

The REMOTE-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 dynamically create messages from templates, defaults, and substituted values of bound variables.

For debugging purposes, a global variable can be set to block the actual transmission of result elements. 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.

D. 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, which can happen when different kinds of problem episodes overlap in real time. 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 (lexical) and MEA (means-ends-analysis) [7], have been extended by a Priority Mode which is orthogonal to 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.

```
(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 all
   {<garbage>                              ; garbage as an atomic
      [List of WME 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 3. Three Rules Illustrating the Collection of Unneeded WME's as an Atomic Action

The priority control mechanism effectively satisfies our real time control needs. It also allows a powerful control over rule interaction between different subdomain areas. The priority mechanism emulates the control aspects of meta rules and eliminates the need for an additional level of indirection caused by their use. (Benjamin and Harrison [17] use meta rules for a different purpose: reasoning about the contents of the conflict set.) Furthermore, it allows rule grouping similar to the use of contexts in EMYCIN [5] and rule-groups in EXPERT [18].

E. Requirements for Continuous Operation

There are at least three basic requirements to operate in a continuous mode. They are:

a) The inference engine should not terminate when no rule is eligible to fire. We implemented a LISP function OPS-WAIT which puts the system into a suspended waiting mode. Any external message (including a timer event) causes the system to resume, with the new data added to working memory.

b) 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 issue an automatic restart instruction 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.

c) 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 unintentionally trigger rules which might be satisfied when only a partial set of working memory elements has 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 following three rule example. (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 WME during initialization.

The final rule in Figure 3 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.

V BUILDING THE KNOWLEDGE BASE

Most of the expertise is encoded in over 500 OPS5 rules distributed between the expert virtual machine and the display control virtual machine. (The rule coding process was facilitated by a programming environment in which a locally developed LISP system [19], on which OPS5 was built, and the system editor XEDIT [20] exist as co-routines.) The expertise was gathered mostly from the operations staff at Yorktown. In addition, systems programmers, manuals and even the designers of the MVS operating system were consulted.

Some of the expertise was encoded in relational tables. (An OPS5 WME is equivalent to a row of a relational data base table. Disjoint cases can be represented as table entries used by driver rules rather than listing separate rules for each case. Use of WMEs as part of the permanent knowledge base has been found to provide a cleaner and more understandable representation in certain cases [21].) Some expertise was implemented in the MCCF translation tables for more direct execution. Also, there are a few parameters hidden from the inference process, that have to do with the MVS interface. Therefore, the knowledge base is not restricted to the rule base alone.

```
(p stop-reception
   (Task  ↑task-id jes-q-space)            ; If the task of
   (JES-Q ↑mode panic)                     ; maintaining JES-q-space
   {<the-Link>(Link  ↑id <L-id>            ; is active, the space
          ↑status <<active i/o-active>>    ; is critically low, and
          ↑receive yes)}                   ; there is an active
→
   (Call remote-make                       ; receiving Link,
      Link-command  ↑id <L-id>             ; then cut the Link
                    ↑receive no            ; and mark the Link
                    ↑rm-to: MCCF)          ; reception status as
   (Modify <the-Link> ↑receive to-be-no))  ; about to be no.


(p start-reception                         ; If the task of maintaining
   (Task  ↑task-id jes-q-space)            ; JES-q-space is active,
   (JES-Q  ↑mode  <>  ¬panic)              ; the space is not
   {<the-Link>(Link  ↑id <L-id>            ; critically low, and
          ↑status <<active i/o-active>>    ; there is an active Link
          ↑receive no)}                    ; not receiving,
→
   (Call remote-make                       ; then
          Link-command ↑id <L-id>          ; reopen the Link
                       ↑receive yes        ; and mark the Link status
                       ↑rm-to: MCCF)       ; as about to be yes
   (Modify <the-Link>   ↑receive to-be-yes))
```

Figure 4. Two Rules from the JES Queue Space Subdomain

The number of rules generally increased along with the coverage. However, increased understanding of the domain sometimes permitted significant reductions by the use of tables and improved knowledge representation in general.

Figure 4 is an example of a pair of rules that stop reception on an incoming link when JES queue space is critically low and restart the reception when it improves. Notice that the Link status value is modified to an anticipated value, awaiting further confirmation from MVS that action has been taken.

A real system must be verified by actual on-line testing. We have found many important pieces of knowledge during on-line testing that the experts did not mention to us. There are other problems as well. Some of the error handling rules can only be exercised during on-line testing if someone sabotages MVS to cause the error. This was done to some extent off prime shift hours. The situation stemming from this real system is that one cannot use a record of test cases (due to dynamic interaction), or use a simulator of MVS (too complex and too large). In contrast, REACTOR [1] was exercised against a simulator and VM [2] used a magnetic tape recording of real time data for a relatively small fixed set of variables. We have used rule walk-throughs, rules to partially simulate some aspects of MVS, and hand interaction in lieu of MVS to aid testing. Thus the validation and certification process is not formal, and needs long experience and a certain amount of confidence derived from seeing the general integrity of system actions.

VI PROJECT STATUS AND FUTURE PLANS

The YES/MVS prototype development took little over one year from inception to on-line testing. We expect the system will be in continuous use at the Yorktown Computing Center by May 1984.

YES/MVS now routinely schedules the queue of large batch jobs. It has alerted MVS operators to network link problems. When jobs which nearly exhaust JES queue space are submitted to MVS, YES/MVS responds with appropriate corrective recommendations to the operator. Other task areas are in the final stages of testing.

Our future plans include broadening the coverage of YES/MVS by the addition of other subdomains, such as facilities to assist the operator during initial-program-loading of MVS, and both planned and emergency shutdown. A learning component is planned for the scheduling of large batch jobs so as to take account of the behavior of previous jobs submitted by a user in scheduling his or her next job. Our success with the computer operations domain causes us to look for the application of expert systems to other areas of computer installation management: capacity planning, configuration and installation.

VII CONCLUSIONS

YES/MVS extends the use of expert systems techniques to continuous, real time, interactive control applications. The extensions we made to OPS5 include facilities essential to such applications and generally applicable to other real time interactive problems such as process control.

We found building the system for actual use to be a challenge involving much more than the usual expert system issues. Integration of the core expert system with a complex real time environment required not only extensions to the OPS5 language, but also some new concerns including how to distribute processing between the expert system and the conventional programming environment. The total system not only interacts intimately with the subject machine, it also interacts with the host system during processing. The difficulty and importance of integration have been observed and emphasized by others [22, 23] but there are still no easy solutions.

While we did learn that OPS5 was an excellent base language, we found that its trigger happy rule firing was

awkward to live with. But it was only through an actual application experience that we uncovered suitable ways to improve this forward-chaining production system language. The techniques we developed are both relevant and effective for real time processing issues. We have also gained many valuable ideas for future improvements in inference engines intended for real time applications.

VII ACKNOWLEDGMENTS

We wish to acknowledge the substantial contribution of Barry Trager, in making the conversion of OPS5 to run under the YKTLISP system on VM/370. We thank the Yorktown Computing Systems management for encouragement during the course of this work and express our deep appreciation to the computer operators who exhibited considerable patience and good cheer during the knowledge acquisition and testing phases of our project.

REFERENCES

[1] William R. Nelson, "REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents", *Proceedings of AAAI-82*, pp. 296-301.

[2] Lawrence M. Fagan, "VM: Representing Time-Dependent Relations in A Medical Setting", PhD Thesis, Stanford University, June 1980.

[3] D. J. Lu, "Watch-Dog Processors and Structural Integrity Checking", *IEEE Trans. on Comput.*, July 1982.

[4] R. Reilly, A. Sutton, R. Nassar, R. Griscom, "Processor Controller for IBM 3081", *IBM Journal of Research and Development*, Vol. 26, No. 1, January 1982, pp. 22-29.

[5] E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, (Elsevier, N.Y.), 1976.

[6] S. M. Weiss, C. A. Kulikowski, S. Amarel, A. Safir, "A Model-Based Method for Computer-Aided Medical Decision-Making", *Artificial Intelligence*, 11:1,2, 1978, pp. 145-172.

[7] C. L. Forgy, "OPS5 User's Manual", CMU-CS-81-135, Dept. of Computer Science, Carnegie-Mellon University, July 1981.

[8] John McDermott, "R1: A Rule Based Configurer of Computer Systems", *Artificial Intelligence*, 1982, Vol. 19, pp. 39-88.

[9] John McDermott, "XSEL: A Computer Sales Person's Assistant", *Machine Intelligence 10*, J. E. Hayes, D. Michie, and Y-H Pao, eds., J. Wiley and Sons, New York, 1982, pp. 325-337.

[10] John McDermott, "Building Expert Systems", Presented at the 1983 NYU Symposium on Artificial Intelligence Applications for Business, May 1983.

[11] G. T. Versonder, S. J. Stolfo, J. E. Zielinski, F. D. Miller, and D. H. Copp, "ACE: An Expert System for Telephone Cable Maintenance", *Proceedings of IJCAI-83*, pp. 116-121.

[12] M. Schor, "Using Declarative Knowledge Representation Techniques: Implementing Truth Maintenance in OPS5", IBM Research Report RC 10455, Yorktown Heights, NY, April 4, 1984.

[13] A. A. Guido, "Unattended Automated DP Center Operation Is It Achievable?", *European GUIDE Proceedings*, June 7-10, 1983, Lyon, France, pp. 440-446.

[14] *The VM/SP System Product Interpreter Reference (Release 3)*, SC24-5239, IBM Corporation, September 1983.

[15] Elizabeth Allen, University of Maryland, "YAPS: A Production Rule System Meets Objects", *Proceedings of AAAI-83*, pp. 5-7.

[16] C. L. Forgy, "OPS-83 User's Manual", in preparation, as a Dept. of Comp. Sci. Report, Carnegie-Mellon University

[17] D. P. Benjamin and Malcolm C. Harrison, "A Production System for Learning Plans From Expert", *Proceedings of AAAI-83*, pp. 22-26.

[18] S. M. Weiss and C. A. Kulikowski, "EXPERT: A System for Developing Consultation Models", *Proceedings of IJCAI-79*, pp. 942-947.

[19] C. N. Alberga et al., "A Program Development Environment", *IBM Journal of Research and Development*, Vol. 28, No. 1, January 1984, pp. 60-73.

[20] *The VM/SP System Product Editor Command and Macro Reference (Release 3)*, SC24-5221-2, IBM Corporation, September 1983.

[21] A. Pasik and M. Schor, "Table Driven Rules in Expert Systems", *SIGART Newsletter*, No. 87, January 1984.

[22] R. Davis, H. Austin, I. Carlbom, B. Frawley, P. Pruchnik, R. Sneiderman, J. A. Gilreath, "The DIPMETER ADVISOR: Interpretation of Geological Signals", *Proceedings of IJCAI-81*, pp. 846-849.

[23] S. J. Hong, "Knowledge Engineering in Industry", IBM Research Report RC 10330, Yorktown Heights, NY, January 12, 1984; also in *Proceedings of Japan Systems Science Symposium*, January 1984.