Expert Operator: Deploying YES/MVS II

R. A. Chekaluk[1], A. J. Finkel[2], E. M. Hufziger[1], K. R. Milliken[2], N. B. Waite[2]

[1]GM Research Laboratories
General Motors Corporation
Warren, Michigan

[2]Thomas J. Watson Research Center
IBM Corporation
Yorktown Heights, New York

Abstract

This paper reports on progress in automating the operation and control of large MVS-based computing installations. Automating operations is a problem appropriate for the application of embedded, real-time, expert systems techniques. The long-term goal is to ease the effort required to automate at many computing installations with different configurations, workloads, and operation policies. YES/MVS I showed that expert systems techniques were advantageous for automating operations. YES/MVS II, reported here, is a collection of components, including an architecture, to assist with automation at many sites. Expert Operator, also reported here, is a system based on YES/MVS II and currently in production use at a General Motors computing center. Expert Operator, then, is a first deployment of the components in YES/MVS II and first real test of their value. Based on this first experiment, YES/MVS II appears to be successful. Major productivity improvements were achieved in developing Expert Operator as compared with YES/MVS I. Expert Operator is well-organized, easily understood, robust to changes, easily expanded, and solves challenging real problems. YES/MVS II and Expert Operator, represent significant progress in automated operations. The ideas reported here should have value in the real-time control of other complex systems.

Introduction

This paper presents the results of research on the use of expert systems techniques for automating the operation of large MVS computer installations.

MVS is IBM's primary operating system for production data processing on large systems. YES/MVS (Yorktown Expert System/MVS Manager) is an experimental expert system for automating the operation of large MVS computer installations. YES/MVS I, the first version, was used regularly in the computer center at IBM's Watson Research Center (Ennis et al. 1986). YES/MVS II is an experimental collection of components that followed from the lessons of YES/MVS I. YES/MVS II includes support for communication with target computing resources, support for a model reflecting the state of target resources, and an architecture used to structure, modularize, and coordinate automation applications. These components are intended to ease the effort required to implement automation at many different installations.

Expert Operator is an expert system based on YES/MVS II and implemented and placed into production use at General Motors Research (GMR). Expert Operator is the first deployment and test of the combined components in YES/MVS II.

The Challenge of Large System Operations

The challenge of operation and management of large MVS installations and networks has been described at length elsewhere, for example in (Milliken et al. 1986) and (Mathonet, Cotthem & Vanryckeghem 1987). Operating large computing systems and networks is similar to automatically controlling many other processes (Chester, Lamb & Dhurjati 1984, Nelson 1982, Fagan 1980, Klein & Finin 1987), but the sources of the unique challenge in operating computing and communication resources are the large scale of such installations and the needs for high performance and high reliability.

Due to the numbers of processors, devices, communication links, and software subsystems, due to the volume and variety of work done, due to interactions in the work, and due to requirements for short response times and almost continuous availability, large MVS installations are *complex*. Console operations plays an important role in both performance and reliability in a large installation. Hence, console operations is a challenging problem in real-time control of a large and complex system.

Due to the important role of console operations, the desire is to automate the function to ensure fast and accurate response according to installation policy.

In spite of the complexity, it is possible to automate console operations at a particular installation. However, designing one software facility in advance that will automate console operations at many sites remains difficult for the following reasons:

1. Different sites have very different configurations, workloads, and policies.

2. At a given site, the configuration, workload, and policy change with time.

The goal of YES/MVS II is to provide components that will reduce the effort required both to automate at many installations and to maintain and enhance an automatic operator at one site.

Background

YES/MVS I. In 1982 a research project at the Watson Research Center was started to investigate the use of expert systems techniques as a base for automated operations. The result was YES/MVS I (Ennis et al. 1986) developed in OPS5 (Forgy 1981).

YES/MVS I established the feasibility of automating complicated, operator actions such as resource management, problem diagnosis and recovery using rule-based programming techniques. However, it became clear that improved tools and structure could significantly enhance productivity in developing facilities that automated operator actions.

General Motors Research. Several years ago at GMR, the requirements of MVS console operations had begun to challenge the human operators. The data center had grown to several MVS systems; the individual systems had grown more complex; the systems had significant interactions; and the console message traffic had grown to over 60,000 messages per day.

The first effort to assist human operators was based on writing a library of short programs in an interpretative procedural language called CLIST (Command List). The CLIST programs (working with the NCCF and NetView subsystems of MVS) provided significant relief. (See Chekaluk & Hufziger 1987.)

This effort, however, soon reached practical limits. Basically, in the CLIST environment, a particular console message triggered a particular CLIST routine from the library. That CLIST routine executed and 'handled' the message. For responding in fairly simple ways to just one or a few messages, this approach proved reasonable. Difficulty was encountered when the operational task to be performed required multi-step actions and knowledge of the current global status of the installation. This experience led to the exploration of expert systems techniques and the adoption of YES/MVS II as a base.

Overview of Improvements

The effort to develop Expert Operator using YES/MVS II as a base has established the value of several advances beyond those of YES/MVS I, namely:

1. Architecture — The present work has produced an architecture for the control portion of a system for automated operations, used this architecture in practice, and demonstrated its value.

2. Model and model manager — The work on both YES/MVS I at IBM and also with CLISTs at GMR showed the need for a global model of the status of the target systems. The present research has formulated general principles for such models and, equally important, rules to manage them, used these principles in practice, and demonstrated their value.

3. Communications — Special purpose, data communication support has been developed to provide high level control over message passing between automation and the targets.

4. Expert system shell — Due to the need for improved tools, personnel at IBM's Watson Research Center designed and implemented the shell YES/L1 or Yorktown Expert System/Language One (Cruise et al. 1987). YES/L1 is a data-driven derivative of OPS5 and includes PL/I as a procedural subset. YES/L1 provides the real-time constructs and good performance needed for automated operations. YES/L1 is the prototype of IBM's KnowledgeTool product.

YES/MVS II includes the infrastructure for all these capabilities, but the value of the work is established by the success of Expert Operator. Similar facilities and structure should be applicable to the real-time control of many complex systems, since the basic ideas are not dependent upon MVS or the automation of operations.

Of the four advances listed above, this paper concentrates on the first two, the architecture and the model and model manager.

Overview of Expert Operator

Expert Operator should be regarded as an *embedded* expert system since it runs continuously in an MVS address space, acts in real-time, and executes via cross-memory communications with NetView, a subsystem of MVS.

Internal Structure

Expert Operator is organized into the following components:

1. Communications support for the exchange of status information and of control commands with targets.
2. A status model that maintains a summary in data of the current state of target resources.
3. The model manager, a collection of rules and routines that establishes and maintains the (status) model. In Expert Operator, the model manager is the largest functional component measured by code volume.
4. Control function, a collection of rules and routines that implement operational policy. This part is organized according to a control architecture based on *problems*, corrective actions or *solutions*, and a technique for selecting which solution to attempt in a given situation.
5. Support for interaction with human operators. This part is the simplest of the functional components and is based on NetView panel services.

Communications

The data sent by target systems is in the form of *messages*. Each message contains data that identifies the target system that generated the message and shows the time the message was generated. Most messages have a *message identifier* that shows the kind of message. Some messages consist of only a single line. Others consist of multiple lines, and the exact number of lines may not be known in advance.

The operator, human or automatic, sends the target systems *queries* and *commands*. Queries merely request information from the target systems. Commands ask that action be taken that will affect the target system. Both queries and commands cause responses (messages) to be generated and returned by MVS.

Expert Operator includes facilities to receive messages from target systems, to recognize, filter, and parse messages, and to build and submit queries and commands.

The Model Manager

History

In YES/MVS I, a collection of rules dedicated to a single type of problem issued its own queries to the target systems (to obtain status data) and issued its own commands to the target systems (to take actions). Experience showed that this approach to target interactions caused problems in two respects: First, the rules and routines of the separate problems largely duplicated the function of interacting with the targets. Second, without a consistent view of target resources, attempts to concurrently address several problems could generate conflicting actions.

So YES/MVS II was designed to include a central, consistent model of the status of target systems and a collection of rules to manage and maintain the currency of this model. The model manager in Expert Operator is a deployment of the YES/MVS II model manager but has been modified and augmented to maintain a model of the hardware and software configuration at GMR.

Model Characteristics

The contents of a model vary over time with the hardware and software configuration of target resources. However, model changes beyond those driven by configuration changes are much less frequent than are changes in policy-dependent logic for handling problems and allocating resources. A functioning model manager that maintains a broadly applicable model can be used by rules that automate a wide variety of operator activities. With such a model in place, the volume of rules that must be written to solve a problem is typically reduced by at least half.

Data in the model is maintained at the granularity that would normally be of interest to a human operator. For example, the current status of individual devices is maintained, but summaries are maintained for the total number of print lines for small jobs in a printer queue (rather than keeping data on individual, small, print jobs). For some resources, MVS volunteers (generates) messages indicating all significant state changes. For other resources, fresh status information must be requested periodically with the period depending on the volatility of the data. Still other information is only collected on demand.

Model Manager Organization

When messages are received, the source is identified, and the time is recorded. If a message is uninteresting, then it is discarded. Interesting messages are parsed, and the status information they contain is used to update the model.

The relation between message identifiers and classes in the model is many-to-many: a given message might update several classes, and a given class in the model might be updated upon the receipt of any one of several types of messages.

All queries and commands to the target are issued through the model manager. Control function interacts only with the model and never interacts directly with the targets.

Each query or command to be issued must be planned in advance and explicitly supported. Every planned query and every command is described in its own query block. A query block anchors a linked list of message identifiers of all messages that could be generated by the query.

Any rule or routine in the control part of Expert Operator can request any query. A query can be requested for immediate execution, for delayed execution, or for periodic execution. A request for periodic execution must be accompanied by a starting time and a period.

Model Integrity. The crucial issues in the model and the model manager are the integrity and currency of data in the model. Currency for a resource is maintained by one of three approaches:

1. Frequent, periodic queries.
2. Queries upon demand, before each reference to the data by control code.
3. Capturing and recording all state changes as those changes are broadcast in MVS generated messages.

Achieving integrity involves several considerations including the following:

If a query (or command) has been issued, but the response messages have not arrived and been processed, then the query is said to be *active*. Since the status model may be in an incoherent state when a response is only partially processed, data in the model that are potentially affected by a query are marked as invalid while the query is active.

For many queries, when the query has been issued but no response has been received, repeating the query could be harmful. The model manager either serializes such queries, or it recognizes that the response from the first query will suffice for the second query.

The meaning of a response can be ambiguous unless the query is also known, so the model manager does not update the model based on responses that the model did not request. However, some important messages are generated by MVS upon the detection of events by MVS. These are recorded in the model whenever they are received.

A query that remains active for an unusually long time can be a symptom of a problem with the target systems. To detect this situation, a rule is written to fire whenever a query has been active too long. The record of active queries must be cleaned-up by this rule. (This concern is similar to the situations reported in Fox, Lowenfeld, & Kleinosky 1983).

Control Architecture

While the model manager is the largest component in Expert Operator, the control component is more likely to change with changes in operational policy, and its organization is critical in simplifying the development of automation applications.

The goals in adopting an architecture for control include the following:

1. A modular framework can organize and compartmentalize the development of rules and routines to solve a new operational problem.
2. The real-time environment of Expert Operator means that resolutions of several problems are often in progress at once. The architecture should provide a framework so that processing for multiple problems can be interleaved.
3. An architecture can organize the resolution of conflicts encountered in the concurrent solution of different problems.
4. Partitioning Expert Operator by function, into modular components with clean interfaces, makes the facility easier to understand, enhance, and maintain.

Solving One Problem

Consider an experienced operator working on just one problem. Some approximation to the following steps would be taken:

1. Detect an exceptional condition in the target systems, that is, a problem.
2. Make initial hypotheses about the problem.
3. Query the target systems for more information if necessary.
4. Revise hypotheses based on new information.
5. Attempt a solution.
6. If this attempt did not solve the problem, return to step 3.

For work on a single problem, we want our architecture to conform essentially to these six steps. In particular, the central part of the architecture is based on *problems* and *solutions*.

Problems and Solutions

Whenever a problem is detected in a target system, Expert Operator creates an instance of class **PROBLEM**. This instance is deleted when the problem is known to have been solved. The **PROBLEM** instance serves as an anchor for all other data relevant to solving that problem.

A *solution* is a planned course of action intended to solve a problem. Such a course of action need not consist merely of a command or two to the targets. Instead, such a course of action can be of arbitrary complexity. In particular, a course of action may involve examination of data in the model, requests for various queries to update the data in the model, additional inferences from the results of these queries, commands to the targets, more queries to follow-up on the results of the commands, etc.

For each solution, there is a class of instances. The existence of an instance in that class records the applicability of that solution to an existing problem. The course of action that is a solution is encoded in rules that are enabled by an eligible instance of the solution class. A field on the solution instance records the eligibility of the instance.

A solution instance is always linked to the problem instance for which it records applicability. When a problem is detected and a problem instance is created, the detection process initiates the selection of applicable solutions and the creation of corresponding solution instances. Several solution instances may be associated with a problem instance. Each such solution instance is initially marked as ineligible to solve the problem.

Selecting and Enabling a Solution

Each solution has three ratings as follows:

- problem severity, i.e., the severity of problems for which this solution is appropriate,
- expected effectiveness of the solution,
- impact, i.e., a measure of the disruption caused in the target system by applying this solution.

The values of these ratings are recorded in the member of the solution class and can be adjusted dynamically.

Due to the variety of possible situations to which one solution could apply, and due to the potential for dynamic change to solution ratings, the choice is made only after a problem occurs as to what solution to apply. A particular rule, which is named the *meta-rule*, makes the selection and sets the selected solution to execute. From among all solutions instances created but not yet eligible, the meta-rule selects the solution that has the highest problem severity rating, breaks ties by selecting the solution with the highest effectiveness rating, and breaks ties by selecting the solution with the lowest impact rating. Having selected a solution in this manner, the meta-rule sets the field in the solution instance that indicates that the solution is eligible. Rules that match against this solution are then enabled to fire and take corrective action.

This approach is similar to that suggested or adopted by other developers, for example (Kastner 1983), (Slagle & Hamburger 1985), and (Klein 1985).

Interaction with Human Operators

A problem can be in one of two modes, *active* or *advisory*. A selected solution inherits the mode, active or advisory, of its problem. In active mode, solutions go ahead and take the actions on the targets as planned. In advisory mode, solutions merely advise the operator about the recommended action to take.

In advisory mode, the actions recommended by a solution are organized into units called *suggestions*. A suggestion is implemented as a linked list of the commands to the targets recommended by the solution. Each recommendation presented to an operator consists of all the commands associated with a solution.

For each problem, solution, or suggestion Expert Operator can have one line of prewritten, parameterizable, English text. An operator explanation facility supports the display of this English text for any instance of a problem, solution, or suggestion. For example, if an operator makes the request

```
EXPLAIN SUGGESTION 12
```

then Expert Operator might respond

```
There are too many jobs on the MVSA
   system
Therefore, I am invoking system cleanup
   routines
The suggestion will invoke a cleanup
   routine for class T output
```

The text is parameterized by the insertion of short character strings typically when the instance of the problem is created.

Rationale

This architecture should be evaluated in terms of the motivations for an architecture listed previously.

Compartmentalization. When one starts to attack a new problem, one already knows that problems, solutions, and suggestions will be the major elements, and each of these three will have one line explanations. Each solution needs three ratings. And, for each solution class, there will be a collection of rules that match against its members. The solution can be debugged in advisory mode. This is enough to constitute a good running-start toward the development of a solution. This same compartmentalization makes activities easier to understand and to maintain.

Interleaved Activities. For the rules in the solutions, the right-hand sides typically are short pieces of code. When a rule has completed firing, the KnowledgeTool inference engine selects another rule to fire, and the next rule can be associated with a completely different activity. So, the KnowledgeTool inference engine acts as a kind of intelligent task dispatcher. (The solution members in working memory act like task control blocks, and the inference engine's conflict set is something like a task ready list.) So work on several concurrent problems is interleaved.

Coordinating Independent Solutions. It is also possible to achieve significant coordination of separately developed solutions: For example, suppose suddenly there is an instance of a very high priority problem. Suppose part of the solution for this problem is to restrict other queries and commands to the targets until this problem is solved. Then, one merely writes a rule subroutine to mark lower priority solutions as ineligible until the high priority problem is solved.

As a second example, suppose a solution is underway, but suddenly new data indicates that another solution procedure is preferable. Then, it is possible to mark the eligible solution as ineligible and the preferable solution as eligible. Entire problems can be permanently eliminated in mid-stream when appropriate in light of new data.

Sample Domains

We briefly describe some of the domains of MVS operations currently addressed by Expert Operator.

Backlogged Jobs

One of the major subsystems of MVS is JES (Job Entry Subsystem). There are two versions, JES2 and JES3. GMR uses JES3, and Expert Operator automates a subset of JES3 operations.

Jobs enter the system from various sources and then work their way through several queues. Each queue is work for a DSP (dynamic support program), a component of JES3. The number of jobs for each DSP can be obtained by means of the JES3 command `8i b`. The following is a sample of output from an `8i b` issued to a JES3 system:

```
8i b

IAT8688 0000(W) 0506(A) OUTSERV
IAT8688 0005(W) 0084(A) MAIN
IAT8688 0000(W) 0002(A) WTR
IAT8688 0000(W) 0001(A) DC
IAT8688 0000(W) 0001(A) RJP
IAT8688 0000(W) 0003(A) INTRDR
IAT8688 0000(W) 0009(A) NJE
IAT8688 0000(W) 0001(A) NJECONS
IAT8688 0000(W) 0001(A) TRIG
IAT8688 0000(W) 0001(A) WATCH
IAT8688 0000(W) 0001(A) WTP
```

These messages may be identified by the `IAT8688` identifier. Each line of output gives data for one DSP, and the name of the DSP is the last token on the line. The numbers show the number of jobs either waiting `(W)` or active `(A)`.

The number of jobs active or waiting in each DSP and the total number of jobs in JES3 should be kept under thresholds. When numbers exceed thresholds, they are said to be *backlogged.*

Expert Operator detects a variety of abnormal situations based on the contents of the DSP queues. Considerations include numbers of enqueued jobs, thresholds, active or waiting, the DSP, the total number of jobs in JES3, etc. Many corrective actions are straightforward, but the variety of possible, abnormal conditions is interesting.

CI Monitoring

One of the most important DSP's is CI (converter-interpreter) that reads the job control language of the job. The CI DSP executes so fast that usually its queue is empty. For example, the command `8i a d=ci` yields

```
8i a d=ci

IAT8520 NO JOBS ACTIVE ON CI
```

However, in unusual situations, for example, a lock is held on a crucial data set, jobs can backlog in the CI queue.

Expert Operator watches the CI queue every three minutes, and it purges and cleans-up after any jobs in CI longer than one minute.

DR Monitoring

A DR (disk reader) is a special type of job in a JES3 system. Installations commonly write DR's for utility or monitoring tasks.

One common technique is to write a DR to run and, then, just before it stops, resubmit itself to run again at some later time. For example, a DR could be written to run once-an-hour on the half-hour, but only on weekdays.

Sometimes a new DR will contain logic errors. One common error causes the DR to resubmit itself as fast as possible. This error can quickly fill JES3 with unwanted jobs and impact the operation of the entire system. For such a problem, fast response is needed from an operator.

Expert Operator monitors the total number of DR's in the system. If that number becomes abnormally large, then repeating DR's are individually identified and abended by Expert Operator.

Development Effort

The work on YES/MVS I consumed approximately fifteen person years of labor. In comparing complexity of function, it is estimated that Expert Operator delivers approximately forty percent of the volume of function in YES/MVS I. However, the work at GMR on Expert Operator consumed about 1.5 persons for about 15 months. Of these 15 months, most of the time was spent on enabling software outside the central structure of control actions. With the enabling software complete, additional control function continues to be added with much less effort.

Due to the feature of advisory and active modes, the system was used in production, in advisory mode almost immediately after rules were written. Production use in full active mode began only a few months after the initial rule development.

Summary

We review the current status of Expert Operator:

- The system is in production use. On weekends and other off-shift times, the system runs unattended.
- All but the most recently developed problem solving rules in the system currently run in active mode, providing explanation to operators only as requested.
- The problem solving function in Expert Operator was developed by a few skilled people in a few man-months of effort. Function in YES/MVS I to solve problems of similar complexity required markedly greater development effort.
- The model manager and control architecture have partitioned and organized the system's rules and routines into components by function and by the type of external changes that require adaptation in the automation facility.
- The individual components of the system have proven to be self-contained, easily understood, and comparatively easy to enhance and maintain.

We and others (Strandberg et al. 1985) have observed that it is often difficult to accurately measure the advantages that accrue to a software assistant. It is nevertheless clear based on the function implemented to date, that Expert Operator has improved the reliability and availability of the MVS production computer systems at GMR, and the demand for attention from the human operators has been reduced. The project has provided an open path to automation of additional aspects of console operations and systems management in the future.

These preliminary results based on one use of the components and architecture of YES/MVS II at GMR indicate that marked improvements have been made in reducing the complexity of function that must be provided by users who want to automate operations at a particular MVS-based computing center.

References

Chekaluk, R., and Hufziger, E., 1987. An Approach to Automated Operation of Complex Computer Systems. Research Publication GMR-5891, General Motors Research Laboratories.

Chester, D., Lamb, D., and Dhurjati, P., 1984. Rule Based Computer Alarm Analysis in Chemical Process Plants. In *IEEE Micro-Delcon 1984: Proceedings, the Delaware Bay Computer Conference.*

Cruise, A., Ennis, R., Finkel, A., Hellerstein, J., Klein, D., Loeb, D., Masullo, M., Milliken, K., Van Woerkom, H., and Waite, N., 1987. YES/L1: Integrating Rule-Based, Procedural, and Real-Time Programming for Industrial Applications. In *Proceedings of Third Conference on Artificial Intelligence Applications.*

Ennis, R., Griesmer, J., Hong, S., Karnaugh, M., Kastner, J., Klein, D., Milliken, K., Schor, M., and Van Woerkom, H., 1986. A Continuous Real-Time Expert System for Computer Operations. *IBM Journal of Research and Development,* Volume 30, Number 1: 14-28,

Fagan, L., 1980. VM: Representing Time-Dependent Relations in A Medical Setting. Ph.D. Diss., Stanford University.

Forgy C., 1981. OPS5 User's Manual. CMU-CS-81-135, Department of Computer Science, Carnegie-Mellon University.

Fox, M., Lowenfeld, S., and Kleinosky, P., 1983. Techniques for Sensor-based Diagnosis. In *Proceedings of IJCAI 1983.*

Griesmer, J., Hong, S., Karnaugh, M., Kastner, J., Schor, M., Ennis, R., Klein, D., Milliken, K., and Van Woerkom, H., 1984. YES/MVS: A Continuous Real Time Expert System. In *Proceedings of AAAI 1984.*

Kastner, J., 1983. Strategies for Expert Consultation in Therapy Planning. Ph.D. Diss., Rutgers Univeristy.

Klein, D., 1985. An Expert System Approach to Realtime, Active Management of a Target Resource. MBA/MSE Thesis, University of Pennsylvania.

Klein, D., and Finin, T., 1985. On Requirements of Active Expert Systems. In *Proceedings of AVIGNON-87, Seventh International Conference on Expert Systems and Their Applications.*

Mathonet, R., Van Cotthem, H., and Vanryckeghem, L., 1987. DANTES: An Expert System for Real_Time Network Troubleshooting. In *Proceedings of IJCAI 1987*, 527-530.

Milliken, K., Cruise, A., Ennis, R., Finkel, A., Hellerstein, J., Loeb, D., Klein, D., Masullo, M., Van Woerkom, H., and Waite, N., 1986. YES/MVS and the Automation of Operations for Large Computer Complexes. *IBM Systems Journal,* Volume 25, Number 2: 159-180.

Nelson, W.R., 1982. REACTOR: An Expert System for Diagnosis and Treatment of Nuclear Reactor Accidents. In *Proceedings of AAAI 1982.*

Slagle, J., and Hamburger, H., 1985. An Expert System for a Resource Allocation Problem. *Communications of the ACM*, Volume 28, Number 9.

Strandberg, C., Abramovich, I., Mitchell, D., and Prill, K., 1985. PAGE-1: A Troubleshooting Aid for Non-impact Page Printing Systems. In *Proceedings of Second Conference on Artificial Intelligence Applications.*