Model Management and the Automation of Computer System Operations

Allan J. Finkel, Seraphin B. Calo, David A. Klein

IBM Research Division
Thomas J Watson Research Center
P.O. Box 704
Yorktown Heights, New York 10598

Abstract

This paper reports on efforts to develop and manage a model of a real time computer system. This model interacts with both the computer system itself, and with heterogeneous software modules designed to control and manage the computer system.

1. Introduction

The design of automation software to control real time systems provides special challenges to software developers. These systems are characterized by rapidly changing status, and complex interactions. Automation software to control a real time system must be able to model this changing status, and make quick decisions based on this model and local operational policy.

Experience indicates that in very complex process control applications, a dedicated subsystem is needed to maintain such a status-model. This paper reports on efforts to develop and manage a model of a real time computer system. The computer systems of interest are large scale, IBM mainframe computers that run the MVS operating system. The model interacts with both the computer system itself, and with heterogeneous software modules designed to control and manage the computer system. Our work indicates that even in rule based environments, an object-oriented status-model manager provides valuable functionality.

2. Model Management

In process control domains, a status-model is a model of the moment-by-moment status of the system being controlled. This status model provides software automation modules with up to date information necessary for the decision making process.

In relatively simple control domains this status-model may be limited to a small set of control variables that are updated and examined directly by process control software. In fact, an explicit status-model may not even be architected into the software.

The domain of computer systems management cannot be so easily modelled. In this domain, the development of effective software automation modules depends on the development of a good system status model, and an associated model manager.

A model manager for a real time system defines and manages a status-model of that system. In particular a model manager is a facility that provides the following services for software automation modules: Data Representation, Data Management, and Data Sharing.

The model manager must define and provide a standard representation of data of interest to software automation modules. For the domain of computer systems management, this information will include models of such devices as processors, printers, DASD devices, terminals, and controllers.

Data management includes such tasks as ensuring that information stored in a data instance accurately reflects the behavior of the instance's real world counterpart; or that configuration information is up to date and consistent. The model manager serves as a buffer between the automation software and the real time system. Model manager data management routines are responsible for interacting with the host system to obtain fresh status information.

The status of a computer system constantly changes. Jobs are processed, devices are turned on and off, users start and terminate sessions. Data about the status of a computer system has a limited lifetime. Data may be valid for time periods ranging from a few milliseconds to several hours. In almost no case is status data valid past the lifetime of a system shutdown.

The heterogeneous nature of the interactions and the relatively high volume of transactions that must be processed by the model manager make most typical database tools inappropriate for this use.

An essential feature of a model manager is data sharing. The model must provide interfaces, (such as an Applications Programming Interface), to allow various software modules to obtain information from the model, and to alter information in the model.
3. Model Manager I

MODEL MANAGER I was a component of an expert system called YES/MVS I ([1]). However it was not an architected component. In retrospect, we were able to observe that we implicitly provided model management services for YES/MVS I. The lack of an architected status-model led to difficulties. It was the responsibility of each module to create the appropriate data structures and initialize their values. This initialization could involve knowledge of system configuration, or query submission. Each module took full responsibility for updating and maintaining the data structures it defined. MODEL MANAGER I was characterized by duplication of effort and lack of consistency.

MODEL MANAGER I pointed out the importance of providing for an explicit architected status-model, and associated common model management services. An effort was undertaken to design a single architected model manager.

4. Model Manager II

MODEL MANAGER II was an important component of YES/MVS II, a second version of YES/MVS. MODEL MANAGER II could be used to define a standard model of a computer system. It provided facilities so that this model could be enhanced.

MODEL MANAGER II also explicitly provided data management services to ensure data validity, and interfaces to manage interactions with the computer system. It served as a central controller and manager of data. Facilities were provided to allow expert system code to request periodic updating, or on demand updating of model data. MODEL MANAGER II processed these requests, served as an arbiter, and if needed submitted the appropriate queries to the computer system.

MODEL MANAGER II was written in rule based code. A forward chaining production system was used, and MODEL MANAGER II services were accessible only to other rule based modules.

Rules can be very effective in coding an operator's knowledge. However rules are less effective in managing a model of system status. Many rules in MODEL MANAGER II were concerned with the format of computer system messages. In fact, for each computer system message of interest there corresponded one or more rules concerned with parsing and managing that message.

These rules were cumbersome to write and cumbersome to maintain. Far worse, these rules competed with the rules of the expert system software automation code for execution by the inference engine. This competition slowed down the expert system, and impaired its problem solving ability.

Thus, while the data representation, data management and data sharing facilities of MODEL MANAGER II were very desirable, the tight integration of the data management code with the expert system automation code proved undesirable.

5. Model Manager III

Experience with MODEL MANAGER II lead us to the conclusion that the data representation and data management components of a model manager should be separate and distinct from the system automation programs. Furthermore, data management should be more closely tied to the data model than rules allow. An object oriented approach was therefore considered more appropriate; i.e., a methodology that would support encapsulation of the data with the knowledge necessary to maintain the data.

While expert system techniques are useful for capturing certain aspects of operational knowledge, they do not provide the best foundation for building a model manager. The working memory of a production system provides little support for the permanence needed in a system model; and, rules are an awkward tool for capturing structured information about computer system resources and their interrelationships.

Additionally, it is desirable to have the model manager execute independently of other programs so that there would be no interference with the problem solving ability of the software automation modules. This consideration thus leads to a model manager that is not directly tied to expert systems or any other reasoning technology, and can thus be used with almost any type of software automation code.

This latest realization of a model management facility can therefore be considered to provide a common representation service that can be generally utilized within the computing environment. It thus becomes the focus of integration for control of diverse, interacting management processes.

MODEL MANAGER III presents automation programs with an Applications Programmer Interface (API) that can be used for interactions with the computer system model. Through this API, a data model can be defined, and instances of data created. Unlike earlier model managers, instances of data can be permanently saved.

To obtain status information, an automation program uses the API to query fields of data structures managed by MODEL MANAGER III. Associated with the data are small programs called methods. These methods can be used to maintain the validity and consistency of the model. For example, upon a data access request, a query method could be used to examine the data in a data structure representing the status of a computing device. The method could then determine if the status data was still relevant, i.e., if the data was received sufficiently recently. If not, the method could submit a query to the computer system for more up-to-date information.

6. References