An Architecture for the Development of Real-Time Fault Diagnosis Systems Using Model-Based Reasoning

Gardiner A. Hall, James Schuetzle,
David LaVallee, and Uday Gupta

Loral AeroSys
7375 Executive Place, Suite 101
Seabrook, Maryland 20706
(301) 805-0300

Abstract

This paper presents an architecture for implementing real-time telemetry-based diagnostic systems using model-based reasoning. First, we describe Paragon, a knowledge acquisition tool for offline entry and validation of physical system models. Paragon provides domain experts with a structured editing capability to capture the physical component's structure, behavior, and causal relationships. We next describe the architecture of the run-time diagnostic system. The diagnostic system, written entirely in Ada, uses the behavioral model developed offline by Paragon to simulate expected component states as reflected in the telemetry stream. The diagnostic algorithm traces causal relationships contained within the model to isolate system faults. Since the diagnostic process relies exclusively on the behavioral model and is implemented without the use of heuristic rules, it can be used to isolate unpredicted faults in a wide variety of systems. Finally, we discuss the implementation of a prototype system constructed using this technique for diagnosing faults in a science instrument. The prototype demonstrates the use of model-based reasoning to develop maintainable systems with greater diagnostic capabilities at a lower cost.

I. Introduction

Diagnosing spacecraft faults is a difficult, error-prone, and time-consuming activity. Spacecraft diagnosis is performed by an operations team composed of a large contingent of highly trained people. These people monitor a satellite telemetry stream containing hundreds of system data points. When an anomaly is detected, the operations team analyzes this data with respect to archived historical telemetry data and detailed spacecraft design information. Analyzing such large quantities of data and developing a hypothesis explaining the data is an extremely challenging task. It is not uncommon for satellite anomaly investigations to take several days.

The already difficult chore of satellite fault diagnosis will be even more demanding in the future. Satellites and their instruments will become more sophisticated and complex, raising the complexity of the fault analysis process. Along with increased complexity, future missions are expected to last longer. A mission life measured in terms of decades rather than years, introduces challenges in maintaining the operations team skill level. The desire to support interactive science operations conducted by people external to the control center will further complicate fault diagnosis activities. The operations crew's ability to maintain a current accurate assessment of the spacecraft's state will be

taxed as more people manipulate the spacecraft and its instruments. Due to these increased complexities, the corresponding control centers are apt to be more costly to build, maintain, and operate.

The application of artificial intelligence techniques promises to help alleviate these problems by increasing the level of automation in spacecraft operations. Specifically, improving the automation level of a control center may result in realizing the following benefits:

a. reducing the risk of catastrophic mission failures

b. reducing the cost of control center operations

c. increased spacecraft and instrument utilization

d. increased retention of key operator's skills

e. an ability to "scale up" control centers to handle more complex spacecraft, more spacecraft and instrument activities, and more users without a proportional increase in cost

This paper describes a system that improves the level of automation in a control center by automating a control center's fault detection and isolation activities.

Background

Our approach to providing automated fault diagnosis tools that quickly and accurately find and solve problems is centered on three basic premises. First is the belief that knowledge base construction and maintenance activities are most appropriately performed by domain experts. Second, a fundamental feature of our expert systems is the separation of problem solving from knowledge acquisition. Third, the tools we build reflect the notion that solving different problems requires different problem-solving techniques. The rationale for this design philosophy is documented in [JAW-87]. Figure 1 illustrates the architecture derived from these design principles.

Our first tool, a rule-based expert system, the Ford Lisp Ada Connection (FLAC) described in [JAW-88], includes an offline knowledge acquisition component and an online inference engine. The offline component is an intuitive graphical editing tool that is used directly by the domain expert. It does not require knowledge of AI or expert systems and is easily learned by the domain expert. The rule base is developed as a graph of nodes symbolically depicted as *and/or gates*, as typically seen in CAD systems for integrated circuit design. Once the expert is satisfied with the rule base it is downloaded to the online system. The rule base is loaded into data structures at run time for use by the embedded Ada inference engine.

FLAC successfully demonstrated the feasibility of real-time expert systems. However, the limitations of production rule systems soon became apparent. Fundamental to these systems is the requirement to enumerate explicitly all possible faults. Intuitively, as the complexity of the system increases, it becomes increasingly difficult to predict accurately every possible fault scenario. Another deficiency in the rule based approach is the inability to gracefully solve problems that change over time. One key requirement for a diagnostic system is the capability to reason about temporal and control relationships between attributes of the target system. Developing a rule base that captures and implements rules describing temporal and control relationships is exceedingly difficult and error-prone.

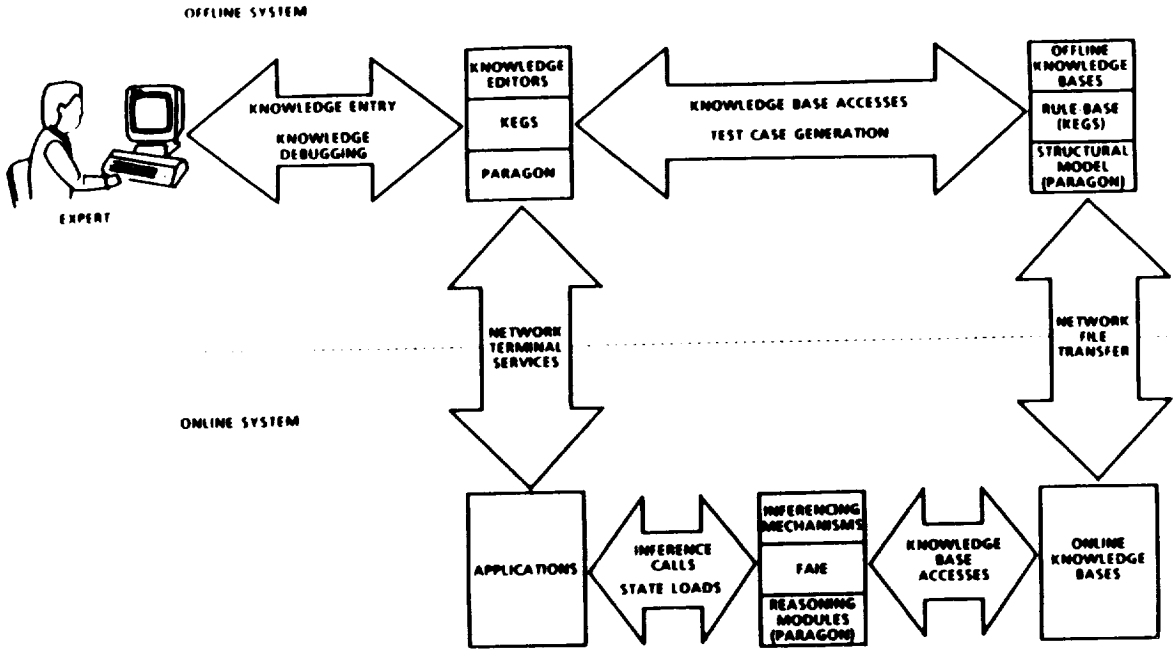

Figure 1. An Architecture for Knowledge-Based Systems

Issues associated with maintaining a rule base large enough to describe a spacecraft also became apparent. Because of the unstructured nature of rule bases, maintenance is difficult when adding or modifying rules. The unstructured nature of rule bases also leads to a formidable verification and validation task. Additionally, as rule bases become larger, maintaining consistency between rules becomes increasingly difficult. Maintaining rule base integrity requires the addition of more rules and routines dedicated to the consistency checking function.

To overcome these difficulties, we began an investigation into a model-based reasoning approach to real-time fault diagnosis. The model-based reasoning approach has promising features relevant to control center fault diagnosis activities. For instance:

a. Model-based systems reason from deeper principles. Model-based systems know the internal processes of a machine and can determine the machinery's state from observed values. In a rule-based system, relationships defining each observation and the machine's state must exist.

b. Model-based systems can reason about a system as it changes over time. Model-based reasoning systems have this capability because events and conditions can be represented by mathematical functions that are close approximations of actual conditions.

Like FLAC, our model-based reasoning system contains an offline graphical component for easy entry of knowledge, and an online embedded diagnostic component. The offline component, originally implemented by Loral's Space and Range Systems division, is a model-building tool called Paragon. Paragon is used to build a structural and functional model of the system to be monitored. The model is exported as a

file to be loaded at run time by the online diagnostic component. The diagnostic system, developed by Loral AeroSys, uses the behavioral model to predict expected states for the system and compares them to actual states as reflected in the telemetry stream. The diagnostic algorithm traces causal relationships described in the model to isolate system faults. The diagnostic system is implemented in Ada and is capable of real-time performance on conventional processors.

The remainder of this paper discusses our experiences with the model-based reasoning approach in more detail. Section II describes the architecture of the system. The implementation of the prototype is covered in Section III, and our results and conclusions are presented in Sections IV and V respectively.

II. Model-based Reasoning System Architecture

In our prototype system there is an offline component for creating and verifying knowledge bases, and an online component for the diagnostic software. This design reflects the architecture of many current control centers (e.g., MSOCC, Space Telescope). The offline systems define telemetry and command databases, while the online systems use these databases for interpreting spacecraft telemetry and building spacecraft commands.

Offline Knowledge Acquisition System

The Paragon knowledge acquisition tool provides a method to construct a detailed structural and functional model of a problem domain. The model is specified in terms of objects, object behaviors, and relationships between objects. These different views of a model can be thought of as defining *conceptual* and *relational* entities. Conceptual entities define concepts existing in the problem domain and are composed of *dynamic* and *static* aspects. Dynamic aspects describe an object's relationships to other objects and how that object may be manipulated. Static aspects describe the object's attributes and how these attributes relate to other concepts. Relational entities describe relationships between two concepts. Each relationship within the model has a specific and well-defined behavior. Figure 2, a screen dump from a Paragon session, provides an example model definition.

Concepts in the Paragon system are either relations, classes, or instances. The Paragon system supports inheritance in the form of **class ⇨ subclass ⇨ instances.** This classification scheme is a strict hierarchy; an instance may have at most one defining class. Using this scheme, a semantic network is constructed representing the real-world system. The frames composing the network are the defined instances. The slots of the frame hold the object's attribute values. Relation objects link the frames to complete the network.

Figure 3 provides an example of applying concepts, relations, and dynamic and static aspects to a physical object, a thermal switch. Two relationships, *temperature* and *current*, affect the concept *thermal switch*. The switch also contains the local attributes *switching temperature* and *output*. The internal process of the thermal switch provides for two possible states: *ON* and *OFF*. The dynamic aspects of the concept of the thermal switch are represented by the links labeled *Temperature >= SW* and *Temperature < SW*. These represent the possible transition conditions between the ON and OFF states. For example, if the incoming temperature value is less than the local attribute switching temperature, control is passed to the ON state. The static aspects of the thermal switch concept are described by the event equations labeled *Output = 0* and *Output = Current*.

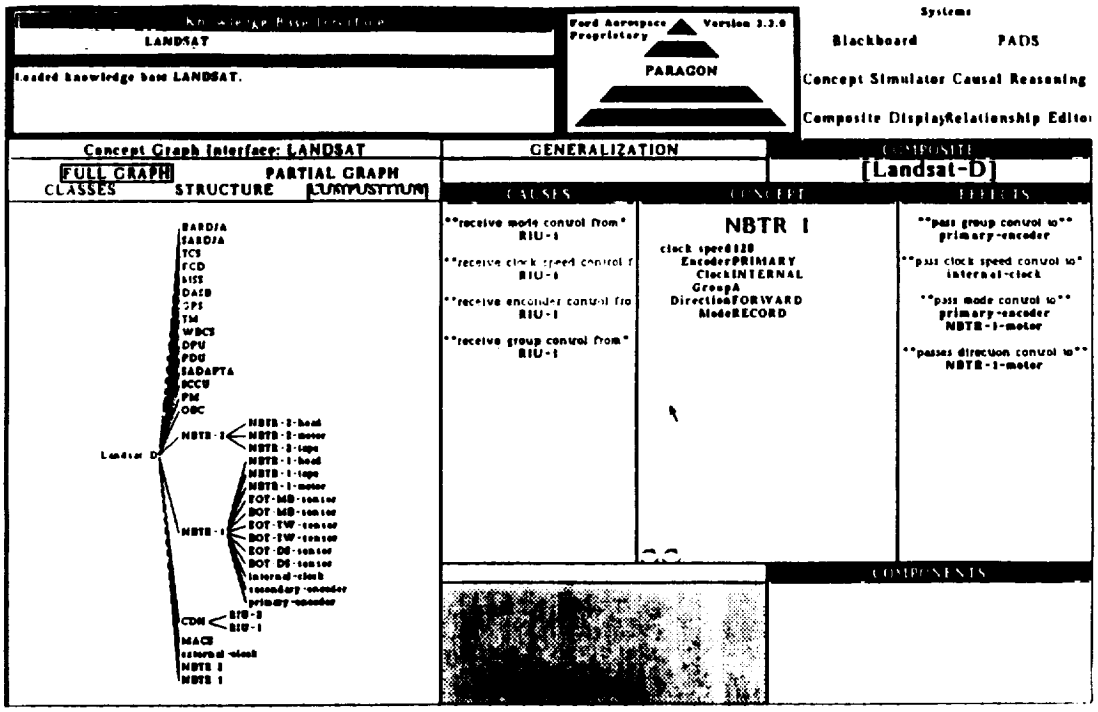

Figure 2. Sample Paragon Knowledge Base

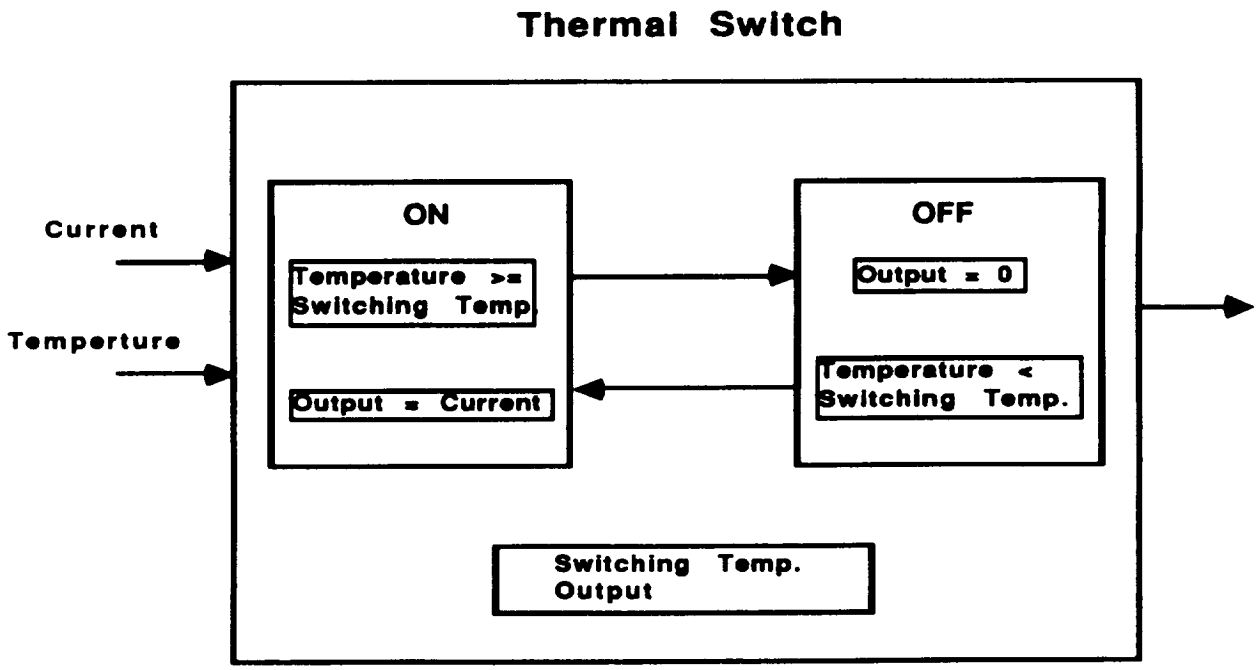

Figure 3. Thermal Switch Class Example

Using these ideas, a model of the physical system is built by recursively defining and instantiating frames and relationships. For a spacecraft, the objects in the system are the onboard components. Each object's attributes are the parameters contained in the spacecraft's telemetry stream. The design and functional information for each object is captured by defining the object's possible states, state transitions, and behavior when in a particular state. An object's attributes may be affected by other local attributes, itself, or attributes of another object. The process of defining objects and relationships continues until a model of satisfactory fidelity is achieved. Included with Paragon are tools for inspecting the classes, objects, and relationships within the system. Also included is a simulation capability for validating the model's correctness.

Online Diagnostic System

The real-time component of the diagnostic system is the Model-Based Reasoning (MBR) module. The primary function of the MBR module is to detect and diagnose electromechanical or other system faults in real time. The diagnostic system is composed of simulation, monitoring, and causal analysis subsystems. The simulation subsystem uses the Paragon-developed knowledge base to generate expected values for each telemetry (attribute) point. The monitoring subsystem synchronizes the simulation with actual time, and performs expected versus observed value comparisons. A mismatch between these two values triggers the causal-analysis subsystem. The causal-analysis subsystem develops hypotheses explaining the observed behavior by examining each faulty component's relationships. These three subsystems work in unison to perform fault detection and diagnosis.

The simulation subsystem uses the information contained in the Paragon model to continuously update the target system's expected state. Specifically, the simulation cycles through all the objects in the system, evaluating each object's state transition criteria for the current state. Once the current state is determined, its attributes are modified to reflect that state. The frequency of the simulation's cycle is the real-time rate. Modifications to the expected state can be effected through external commands, scheduled activities, or the model's internal processes. Maintaining this model provides a reference point for evaluating the spacecraft's health.

The monitoring subsystem is responsible for fault detection. The monitoring process compares time-synchronized, simulation-generated, expected values with actual system-measured values (telemetry in the case of a space system) at predefined time intervals (cycle). A component is considered to be abnormal when these two values disagree. These abnormal components, along with their attributes, actual and expected attribute values, and fault-detection cycle identifiers, are posted to a blackboard structure, called the Abnormal-Components-Blackboard. The Abnormal-Components-Blackboard is inspected to determine if the detected abnormal component exists on the blackboard. If the component does exist on the blackboard, its fault-detection time-cycle identifier is updated with the old fault-detection time-cycle number before it is posted to the blackboard. Whenever abnormal components are detected, further analysis is performed by the causal analysis subsystem to isolate the exact cause(s) of the fault(s) from the abnormal components list.

The causal analysis of suspected abnormal components relies on functional and design information provided by the Paragon model. The basic fault-diagnosis strategy for the causal analysis is:

a. The list of suspected components is read from the Abnormal-Components-Blackboard. A node corresponding to each suspected component is created. These nodes are referred to as Fault Mechanism Nodes (FMN) and are maintained in a list structure.

b. Design and causal link information is obtained for each faulty component.

c. During this step, the causal-effect pointers of the FMNs are assigned. Three types of pointer are set: In-link, Out-link, and Next pointers. In-links point to FMNs whose components affect the attribute(s) of the current FMN. FMN Out-links point to FMN(s) whose component(s) attribute(s) can be affected by the current FMN. The Next pointer simply points to the next FMN. Setting In-link, Out-link, and Next pointers transforms the FMN list into a graph, referred to as a Fault Mechanism Graph (FMG). Figure 4 shows a FMG. Each block contains the component name, In-link, Out-link, and Next FMN pointers. As shown in Figure 4, the component Power Supply 1 has a null In-link pointer indicating that it is not affected by any other FMN. The Out-link pointer of Power Supply 1 points to the node Instrument Power. This indicates that Power Supply 1 causes Instrument Power to be abnormal. Instrument Power's In-link pointer indicates that Instrument Power is affected by Power Supply 1. The component VNIR FPA has a null Out-link pointer indicating that it does not affect other FMNs. These interpretations can be similarly applied to the other nodes of the graph.

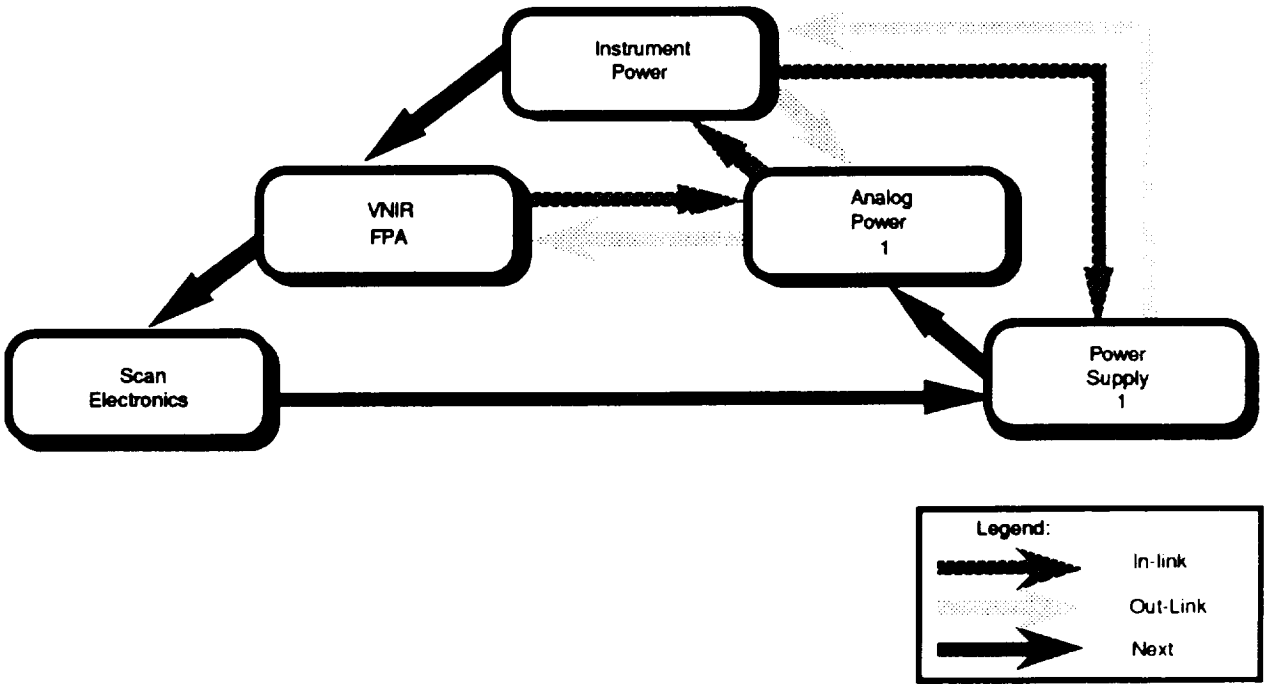

Figure 4. MODIS Fault Mechanism Graph

d. The In-link and Out-link pointers of each node of the FMG are examined. Components with null In-link pointers are considered to be fault sources. Fault-propagation paths are computed by iteratively selecting those FMN's with null In-link pointers, tracing the node's Out-link pointer to the affected FMN and tracing the affected FMN Out-link pointer to other affected FMNs until the current Out-link pointer is null. These paths explain the order in which components became abnormal.

Steps a through d are repeated when a fault-detection cycle detects an abnormal component, or a previously detected abnormal component is found to have returned to a normal state.

III. Prototype Implementation

We demonstrate our model-based approach for real-time fault detection and diagnosis in a testbed environment. The testbed is a complete command and control environment for the Moderate Resolution Imaging Spectrometer (MODIS), a future Earth Observing System (EOS) instrument. Our prototype runs on a VaxStation 3100 and is implemented in the Ada programming language. The MODIS model was developed using the PARAGON tool on a Symbolics 3640 and downloaded to the Vax workstation.

The model, based on a proposed design for the MODIS instrument, took three months to implement using Paragon. The MODIS model consists of over 50 component classes, 80 components, and 11 types of functional relationship. In addition, the model is capable of responding to 96 different instrument commands, and transmits 132 different telemetry points. The model includes definitions for all normal instrument states, state transitions, and internal attribute update equations for all components.

The testbed contains three processors to provide a high-fidelity environment for evaluating control center automation techniques. The architecture of the testbed is shown in Figure 5.

The Symbolics is the offline processor, used for creating knowledge bases. One of the VaxStations is dedicated to control center functions. In addition to fault diagnosis, there is software for:

a. Receiving and decommutating a 2 Kbs stream of packetized telemetry

b. Processing and transmitting instrument commands

c. Displaying graphically instrument telemetry data

The other VaxStation, the telemetry source, executes simulation software generating instrument telemetry. The simulator has the capability to:

a. Modify the value of any object's attributes

b. Update the current state of an object

c. View any object's attribute values

d. Control the length of simulation cycle time (useful for debugging)

e. Accept and process instrument commands sent from the ground

f. Packetize and transmit telemetry.

Telemetry and commands are exchanged between these two processors by way of Ethernet using the TCP/IP protocol.

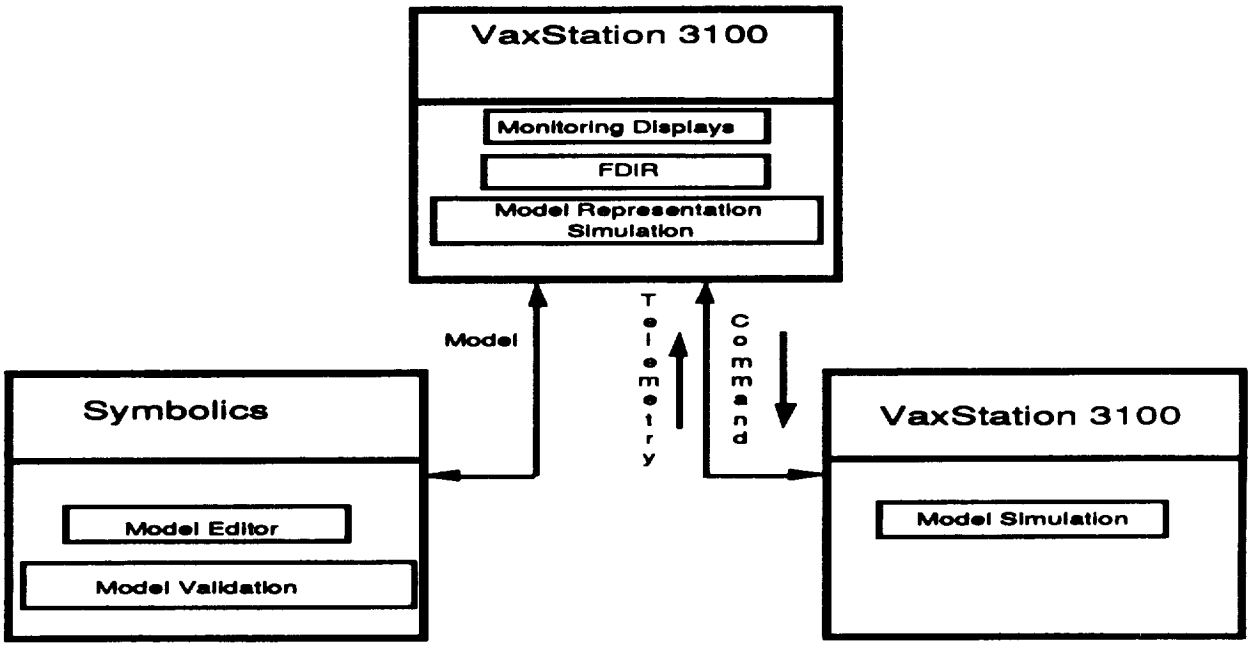

Figure 5. MODIS Control Center Testbed

IV. Results

Using the testbed, the prototype MBR system has been tested under several different fault scenarios. These fault scenarios were developed on the basis of Loral's spacecraft operations experience. The types of fault scenarios tested were:

a. Components that commonly fail during mission (e.g., a sticky relay)

b. Rare or infrequent component faults (e.g., a failed door drive motor)

c. Multiple simultaneous fault scenarios (e.g., a failed heater and a faulty relay).

We also tested the case where two components affecting a common component fail. In this scenario, our prototype identified both components as failed. In all fault scenarios tested, the MBR system accurately detects and isolates the source of fault. Importantly, these fault scenarios were designed after the implementation of the model and fault diagnostic software. In no case were component specific rules describing fault conditions or causes implemented.

V. Conclusions

These preliminary results suggest that model-based reasoning is a viable method for automating spacecraft fault detection and diagnosis activities. On the basis of this research several advantages of the technique are apparent:

a. A model-based system is capable of detecting and diagnosing unpredicted, non-intuitive faults in a continuous, dynamic system in real-time

b. The knowledge acquisition process is simplified

c. The maintenance of the knowledge base is simplified

d. This technique can be leveraged with other control center and spacecraft implementation efforts.

This system has the capability for detecting and diagnosing unpredicted faults. The test cases that have been devised emphasize this point. The design of the fault scenarios used for test purposes was based upon operational requirements rather than diagnostic system capabilities. During demonstrations of the prototype, faults are generated "on the fly" by having members of the audience select the component to be faulted.

Knowledge acquisition activities for implementing the system are reduced. Building the knowledge base for a model-based reasoning system only requires the ability to describe a correctly operating system. Since there is no need to enumerate all possible behaviors, the amount of time required for constructing knowledge bases is reduced. Verification of the knowledge base is easier. Using the physical system as a reference point allows a simple comparison demonstrating the model's accuracy. One of the advantages of representing a physical system as a network of objects is that this presentation lends itself to a graphical representation. A graphical representation is advantageous because it allows the knowledge-base builder to view components from different perspectives.

An important advantage a model-based has over rule-based systems is that knowledge-base maintenance is an easier task. The object orientation of the model simplifies knowledge base maintenance. As each object's interfaces with other objects in the system are clearly defined, modifications can be localized to the object, reducing the potential for harmful side effects. The object-oriented approach also provides for potential knowledge-base reuse. For example, libraries containing generalized reconfigurable objects can be built.

The single most important advantage of a model-based system may be that it is complementary to current control center designs. Calculating the expected state for each on-board component provides a mechanism for dynamically updating telemetry limits and alarm values. Another key point is that most projects construct simulators for ground system verification and training. The model-based technique for fault diagnosis provides a method for leveraging these simulators into day-to-day operations.

References

[JAW87] Jaworski, A., LaVallee, D., Zoch. *A Lisp-Ada Connection,* Proceedings of the 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, (1987).

[JAW88] Jaworski, A., Thompson, J. *Automated Satellite Control in Ada,* Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, (1988).

Bibliography

Items not specifically referenced in the text.

Ferguson, J.C., Siemens, R.W., Wagner, R.E.*STAR-PLAN: A Satellite Anomaly Resolution and Planning System,* proceedings of AAAI Workshop on Coupling Symbolic and Numerical Computing in Expert Systems, (Aug. 1985)..

Fulton, S., Pepe, C. *An Introduction to Model-Based Reasoning*, (January 1990), AI Expert (pp. 48 - 55).