MODEL-BASED REASONING FOR POWER SYSTEM MANAGEMENT USING KATE AND THE SSM/PMAD

A FINAL REPORT

SUBMITTED TO NASA-MSFC IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

OF CONTRACT NAS-NAS39385

By

Robert A. Morris

Avelino J. Gonzalez

Daniel J. Carreira

F.D. Mckenzie

Brian Gann

December 1993

Acronyms Used in this Document

FDIR ..... Fault Detection, Isolation and Recovery
FRAMES ..... Fault Recovery and Management Expert System
IPC ..... Intelligent Power Controller
KATE ..... Knowledge-based Autonomous Test Engineer
LLP ..... Lowest-level Processor
RBI ..... Remote Bus Isolator
RPC ..... Remote Power Controller
SSM/PMAD ..... Space Station Module/ Power Management and Distribution
TTA ..... Time-To-Action

Contents

List of Tables

List of Figures

1 Report Summary

The overall goal of this research effort has been the development of a software system which automates tasks related to monitoring and controlling electrical power distribution in spacecraft electrical power systems. The resulting software system is referred to hereafter as the Intelligent Power Controller, or **IPC**. The specific tasks performed by the IPC include:

1. Continuous monitoring of the flow of power from a source to a set of loads;
2. Fast detection of anomalous behavior indicating a fault to one of the components of the distribution system;
3. Generation of diagnosis (explanation) of anomalous behavior;
4. Isolation of culprit (faulty object) from remainder of system; and
5. Maintenance of flow of power to critical loads and systems (e.g. life-support) despite fault conditions being present (recovery).

The collection of these operations is called FDIR (fault detection, isolation and recovery). The IPC successfully performs each of these operations.

The IPC system has evolved out of KATE (Knowledge-based Autonomous Test Engineer), developed at NASA-KSC. KATE consists of a set of software tools for developing and applying structure and behavior models to FDIR applications. KATE includes an AI system for diagnosis which employs a technique called model-based reasoning. The major impetus for this research is the desire to test the hypothesis that model-based reasoning can be successfully applied to spacecraft power system FDIR. Our review of the research literature on model-based FDIR has produced no evidence of previous efforts yielding results proving or refuting this hypothesis. The significance of this effort, therefore has been the confirmation of the hypothesis that model-based reasoning can be successfully applied in this domain.

Developing the IPC required extensive testing in a real-time environment. To meet this requirement, the Space Station Module/ Power Management and Distribution

system (hereafter, SSM/PMAD or simply PMAD) was utilized. The PMAD breadboard is a distribution system of space station components built to develop and test automation software to be used on Space Station Freedom [10]. The IPC was tested on PMAD using two methods: local and remote. Remote testing consisted of internet transfer of data and commands between the breadboard, located in Huntsville, and the R&D sites, viz., Orlando and Melbourne, FL. Initially, the remote testing was simply a matter of expediency, since it would have been expensive to carry out the research at NASA-MSFC. It soon became clear, however, that the internet delay was an important factor in evaluating the IPC, since it simulated a scenario in which the IPC functions as a ground-based controller of an in-flight spacecraft. Local testing, on the other hand, simulated conditions in which the IPC functions as an on-board controller.

The remainder of this document describes in detail each of the important topics related to the development of the IPC. First, a more extensive introduction to the project goals, methodology, and tools utilized will be presented. There follows a discussion of model-based reasoning as a technique for performing FDIR. The architecture of the IPC itself is discussed in section 6, following by an extensive summary of the tests performed on the PMAD (section 7). Finally, by way of final summary, an analysis of the test results in the context of the proving the feasibility of the concept of model-based power control.

2 Project Requirements and Motivation

There are numerous advantages of automating tasks related to spacecraft power management in general, and control of power distribution in particular. These advantages are particularly dramatic in the case of future, long mission spacecraft such as Space Station Freedom, where it is especially inefficient to maintain continuous, manual monitoring and control of vehicle subsystems. We envision an IPC, in its final form, to reside on board, and be capable of both fully autonomous and interactive decision making. Alternatively, the IPC could reside as part of the ground-based automation software used to assist ground controllers in spacecraft FDIR.

To make significant strides to realize this goal, the IPC developers were faced with a number of initial tasks. The first was to classify the sorts of faults that can occur in power distribution systems. In general, two basic kinds of faults can occur in a power distribution system that affect the ability of the entire system to perform properly. One of them is an open circuit, where the source of power is unintentionally cut off from the load. Such incidents result from inadvertent tripping of a circuit breaker, or from physical damage to a conductor that causes it to lose electrical continuity. Depending on the location in the hierarchy where the open circuit took place, this can disable either only one, or a large number of specific loads.

The second and more significant of the types of faults is a short circuit, which can cause the electric power flow to bypass all or some of the loads and render them useless, even though the short circuit may not have taken place within the affected loads themselves. A short circuit can also have destructive side effects if it causes large amounts of current to flow in the circuit. For this reason, the faulty components must be immediately isolated from the rest of the circuit, even if this means disabling some loads until the condition that caused the short circuit can be eliminated. The goal is to isolate the short circuit while disabling the fewest loads.

Isolation of a fault has to be done quite rapidly in order to avoid the damaging heat buildup that occurs when large currents flow in conductors or equipment not designed to handle them. It is typically desirable to interrupt a fault current (isolate the short circuit) within 0.25 to 0.50 seconds from detection. These numbers are representative, but in general, the larger the current flow, the faster it needs to be interrupted.

Some loads, however, are considered critical in nature, and cannot afford to be isolated from the power source under any circumstances. Examples of these are power to an operating room in a hospital, power to a large computer bank, power to fire control equipment, as well as life support subsystems in spacecraft. For critical loads, redundant sources of power or paths from such sources are generally designed so that upon disability of one source and/or path, the other one is activated immediately to maintain (nearly) uninterrupted power flow. Access to the alternate source of power can be enabled through the closing of normally open circuit breakers, which establish

a path from a power source to the critical load.

Electric power systems are monitored at various locations throughout the network, typically coinciding with the location of a breaker. The monitoring function is carried out with voltage sensors and current sensors (called voltage transformers, or VT's, and current transformers, or CT's, respectively), the latter being more common than the former.

Our primary objective, then, in building the IPC was to automate the tasks related to power distribution FDIR. As a preliminary stage in this process, an extensive review of the research literature describing similar efforts was undertaken. These are summarized in the following section.

3 State of the Art in Autonomous FDIR

The state of the art in power distribution system control can be characterized as employing one or more of the following techniques:

1. Sophisticated switch gear and other devices for fast local response;
2. Global monitoring using conventional computer hardware and software;
3. Limited capabilities for software diagnosis, control and recovery using mathematical modeling or artificial intelligence; and
4. Promising, but on the whole untested, new directions in computer automation using parallel processing and neural networks.

The traditional means of protecting a power system has been local in scope. Upon detection of higher than normal current values from a CT, a breaker will be commanded to trip open and interrupt the flow of electricity to the load(s) downstream from it. The interface between the circuit breaker and the sensor is provided by protective relays which typically possess a mechanism for sending a signal to a nearby switching device to trip itself as a response to an abnormal situation recognized by the sensing device. This is often referred to as local control, because the sensor does

not have any indication of current values at other locations in the network. The tripping of breakers is coordinated through short, pre-determined, time delays to allow the breaker located closest to the short circuit to trip first. Such local control has been the norm for many years in earth-bound power systems due to the absence of a controlling device powerful enough to combine all the inputs and reason about them.

The clear advantage of local control is speed; being local to the device means no overhead is incurred as the result of communication to a global controller. On the other hand, a protection scheme based on local devices is as reliable as the most unreliable device in the system. A malfunction of one of these can lead to the failure of the entire system if a short circuit happens to take place within its zone of protection.

Intelligent global control of an electrical power distribution system, where one decision-making device (a controller) has a global view of all sensor readings, can provide significant advantages in terms of reliability, economy, and ease of reconfiguration over the local means of control. Information about an entire electrical power distribution network provides the capability to recognize and isolate faults in the system with only one monitoring and controlling device. A reliable intelligent power controller, therefore, represents an improvement in the reliability of the monitoring, diagnostic and isolation function for the entire system.

Reliability can also be interpreted as correct action in the presence of potentially incorrect readings (referred to as security). Global control provides the framework for verification of the validity of sensor readings through comparisons with other sensors in different locations in the power system, something that local control is not capable of doing.

Additionally, global control can facilitate the recovery from faults and can reconnect critical loads to an alternate source of power without depending on the reliability of the local relay-type devices, whose failure to recognize the condition can result in serious consequences.

Third, from an economic standpoint, the cost of a single intelligent device is generally lower than that of several local devices. This difference becomes more pronounced for larger systems. Moreover, from the maintenance cost viewpoint, there is no need to perform periodic maintenance on several local devices. This can be

a significant advantage in applications such as manned space vehicles where such maintenance is costly due to the inaccessibility of the devices and the high cost of labor.

Finally, changes to the loads or to the components of the power distribution system are typically the norm during the course of the years of operation of a power system. Such changes must be quickly reflected in the fault detection and isolation schemes in order for them to remain effective. In the case of local control, this may require replacing some devices by new ones which are of a different rating, time delay or even operation mode. Furthermore, since traditional protection in power systems depends on the coordination of devices in different zones of protection, changes in the system configuration may require the modification of devices in other zones in order to maintain coordination. In an intelligent automated global control environment, however, all modifications would be done to the information that the controller has about the system, which can be done much more easily.

The emergence of powerful microprocessors have allowed inexpensive global controllers to be applied to this problem. However, the techniques used for implementing the global control vary and sometimes have significant drawbacks. This next section will discuss them.

3.1 Knowledge-based Approaches to FDIR

Knowledge-based systems have shown significant promise as global control mechanism. Broadly speaking, there are two main approaches to FDIR using artificial intelligence techniques: the *experiential-based* and the *first principles-based* approaches.[1]

The first approach applies associational knowledge based on human experience, captured through various knowledge acquisition techniques. This knowledge can be expressed logically as propositions of the form:

If $\langle symptoms \rangle$ then $\langle fault \rangle$.

[1] A more common terminology for classifying these approaches is *rule-based* vs. *model-based*. This is somewhat misleading, however, since, on the one hand, models based on first principles can be expressed as rules, and, on the other, a set of rules can be said to collectively model a system.

where collectively this knowledge associates one, or a combination of, sensor readings to a malfunction, and can suggest a remedial course of action. Several systems using this approach have been described in the literature, most of them as applications to space power systems [1], [14], [18], [23], [27], [33], [35], [38]. One of the more advanced systems based on associational knowledge is FRAMES (Fault Recovery and Management Expert System), developed in conjunction with the SSM/PMAD at NASA-MSFC. FRAMES is unique in containing as part of its control mechanism a means of managing both the knowledge base itself (through a classification of the possible problems the system can exhibit), and the sensor data that is processed (through a clustering of symptoms).

While the experiential-based approach represents a significant improvement to local monitoring and control, they have been known to suffer from certain drawbacks. Among them is the fact that only faults which have been previously experienced and represented by the knowledge engineer can be successfully identified. This is because, being heuristic-based, the reasoning is based on the past experience of domain experts. If the requisite experience does not exist, or is not represented within the knowledge base due to the uncommon or unexpected nature of the fault, then that fault will not be detected. Furthermore, associative systems can be cumbersome to modify when changes in the system configuration are introduced.

In its original form, the first principles-based approach represents knowledge about a physical system in terms of *structure* and *behavior* under normal operating conditions. Logically, the knowledge can be depicted as propositions of the form

$$\text{If } not - abnormal(o_k) \text{ then } output(o_k) = f(input(o_k))$$

where f expresses knowledge about o_k's behavior as an input-output function. Recent research on diagnostic knowledge-based systems is relying more and more on employing the (non-associative) model-based approach. Since the first principles-based approach was the one taken in the IPC, it will be useful to acquaint the reader with a more extensive overview.

3.2 Model-based Reasoning for FDIR

Recent advances in artificial intelligence for developing reasoners for diagnosing complex systems, such as power systems, stress the need for a robust knowledge representation for the system being diagnosed. A representation based on the strmcture and behavior of each component of the device offers, for many, the best solution to the robustness problem. By structure is meant knowledge of the connectivity of the component to the rest of the system. By behavior, as noted above, is meant roughly how the component transfers (a set of) inputs into (a set of) outputs. Behavioral models can be of two broad kinds: either correct behavior models or faulIt models. As indicated, correct models model the proper functioning of a device, whereas fault models describe common ways in which a component can misbehave. Fault models are considered attractive because they provide a way of incorporating some of the experiential knowledge provided by experts into the knowledge base, and also provide a more detailed explanation of failure than correct models can often provide. Knowledge about structure and behavior can be encapsulated into a object-based framework, where the connectivity of the objects in the framework exactly reflects the connectivity of the objects in the system being modeled.

In the IPC, structure and (normal) behavior models are used by the reasoner to simulate the performance of the actual system. Inputs are fed into the model which correspond to the inputs to the actual system. These values are propagated, using structural and behavioral knowledge, throughout the model to a set of outputs, which correspond to sensor readings at those points. By this method, the system can predict these readings based on knowledge of the inputs.

The ability to predict using structure and behavior knowledge is the basis of *model based diagnosis.* More specifically, the discrepancies between observed and predicted values drive the diagnostic reasoner. The reasoner attempts to find the smallest set of components whose failure would explain the discrepancies between prediction and observation. Logically, the problem is to maintain the consistency of the knowledge, including the current observations. The computational complexity of maintaining the consistency of knowledge is, in the worst case, not something any algorithm can do (i.e., the problem is undecidable), and in general a difficult computational problem;

hence, it is important for automated reasoners to have a mechanism for guiding and controlling the search for a solution.

In general, the model-based diagnostic process is often viewed as having three parts: first, generating a set of possible suspects, testing each suspect, either individually or in sets, and discriminating among the suspects that remain after the final test. The generation phase uses the structural knowledge about the system to collect components that might have caused the discrepancy. A variation of this method employs fault models of each suspect which can be applied by the simulator to see if the faulty behavior can be reproduced. The discrimination phase often involves performing additional measurements to further reduce the suspect list.

One approach (constraint suspension [5]) for combining generation and test is to view each component's normal behavior as setting constraints on the behavior of the entire system. Then abnormal behavior (i.e., when one or more component is a suspect) is the case where the component's constraints on the whole system is unknown. This case can be simulated by removing the constraint knowledge from the knowledge base; if the inconsistency in the knowledge is thereby removed, that component is the sole cause of the failure.

A diagnosis can be viewed as a set of components which explain all the conflicts in the knowledge, where a conflict is a set of components at least one of which must be malfunctioning, given the knowledge. One common preference criteria for ranking diagnoses, hence improving the search space, is in terms of mininality: this says that the most common failures are to a small number of components. A special case of this is the assumption of a single point of failure. Other, more sophisticated techniques for ranking diagnoses use probabilities. Another technique employed to improve effciency in reasoning is the use of truth maintenance [6]. Briefly, this method involves recording all the conclusion drawn from the model and observations, to be reused, without computational cost, in future inferences.

The operation consuming the most computational resources while performing model-based diagnosis is the use of the model for propagation, either forward for prediction, or "backward" (i.e. from effect to cause) for the purpose of ditgnosis.

Techniques like truth maintenance, probabilistic ranking of diagnoses, and fault models, have arisen as a response to this complexity. The system's behavior is measured mathematically in terms of the algebraic expressions describing its behavior. Current constraint oriented model-based diagnostic systems, for example, are "local" in the sense that they propagate values through one model component at a time, solving at each stage one equation in one unknown [5]. More complex systems may have behavior whose algebraic representation consists of equations with more than one unknown. This corresponds in the physical world to more complex causal dependency among the components; e.g. mutual dependency or reconvergent fanout (a signal that branches and then reconverges at a later point).

Researchers in model-based reasoning have attempted to solve the issue of complexity in a number of ways. The most promising, in our minds, is based on the idea that reasoning in the face of system complexity requires the ability to perform *abstractions* in order to guide the reasoning process [37]. Informally, abstraction is the process of focusing on only what is essential, ignoring what is inessential. Two kinds of abstraction, behavioral and structural, are possible. Behavioral abstraction ignores certain characteristics components have and focuses on only ones deemed important. Structural abstraction ignores details related to connectivity of objects, focusing only on a subset of these connections. Current research views abstraction as occurring dynamically, when needed, which implies the presence of multiple models of a system which differ in their levels of abstraction. The reasoner is faced with choosing a model for prediction/diagnosis, and well as applying the model.

Every model-based diagnostic reasoner possesses three high-level modules in its architecture. These are:

- a predictor which generates behavioral predictions based on the model, and detects discrepancies between observed and predicted behavior;
- a candidate proposer which generates conflicts from these discrepancies; generates candidates based on the conflicts; and discriminates and refines candidates; and
- a diagnostic strategist which controls the diagnostic process, in general, by

determining the next step in the process of generating a diagnosis [24].

Predictors typically employ constraint propagation [7], which is the process of computing the deductive closure of the model's knowledge, given a set of inputs. The task of candidate generation in model-based diagnosis, in systems like GDE [7], views the computational problem as one of maintaining the consistency of knowledge about the system.

The discrepancies between observed and predicted values drive the diagnostic reasoner. When discrepancies emerge, an inconsistency between predicted and actual values results. The system records the dependencies of the behavioral predictions made by the model of normal behavior, and determines which assumptions have led to the inconsistency. The reasoner does this by attempting to find the smallest set of components whose failure would explain the discrepancy between prediction and observation. Removing correctness assumptions will eventually make the knowledge consistent, and the result is a set of candidates, or hypotheses, for explaining the discrepancies.

The traditional diagnostic strategy adopted for controlling the process has been the so-called *dependency-recording strategy* [24]. In this approach, the system records the dependencies of the predictions in order to determine which of the assumptions used in modeling the system have led to conflicts. An Assumption-based Truth Maintenance system (ATMS) has been employed for this bookkeeping operation, e.g. as part of the GDE system. The other main approach to strategy selection has been the iterative search strategy [25]. This involves using the discrepancies to search through a space of possible variations from the normal model, until a matching fault model has been obtained.

There are two major obstacles in developing and applying structure and behavior models for reasoning about complex systems for diagnosis: limiting the amount of computation required to reach a diagnosis, and building a model of the system of sufficient detail (granularity) to be useful in diagnosis. One complicating factor is that meeting one of these two requirements tends to inhibit accomplishing the other. Solving these dual problems constitute open research topics in the field of model-based diagnosis. Systems have been developed which incorporate some of the potential

solutions to the problems, but few, if any, systems have progressed to the deployment stage. These problems are illustrated in the next section in association with the development of IPC models.

3.2.1 Structure and Behavior Models For Power Distribution

The structure of a system of components of a power distribution system can be represented in a straightforward manner. Based on the granularity desired in the model, busses, wires, loads, switches, loads, batteries, and power sources of various kinds can be represented as objects in a model. Connectivity in structure and behavior models is depicted logically as statements of the form

$$output(o_1) = input(o_2).$$

This statement signifies that o_1 is connected upstream to o_2. In a power distribution model, (more specifically, to model what is been termed secondary power distribution) it is essential to include representations for the switches, buses, and loads. More granular models would include wires or cables, but for our purposes this was not required. It is also required to model sensors (as components carrying output of the system) and interfaces with the harware used to control the objects. These are called in the IPC model *commands*, representing inputs to the system.

The correct behavior of a set of power system components is characterixed by a set of first principles which describe the correct functioning of the system. These consist of rules which characterize the voltages and currents presented in the system circuit. Voltages and currents are determined by considering the constraints imposed by the behavior of the components, as well as the constraints imposed by the connectivity of the components. Constraints imposed by the interconnections of the circuit are expressed by Kirchoff's voltage law (KVL) and Kirchoff's current law (KCL). These laws are best viewed as applied to the entire system, or, as an abstraction, to a graph of the system. The behavior of a circuit can be characterized as a set of impedance equations and the equations implied by KCL or KVL. In this manner, KCL and KVL provide the constraints of interconnection.

As noted, model-based diagnostic systems have modeled the functionality of a device as a local transfer function that relates its input to its output. This is reasonable in digital electronic systems, and process systems can be approximated similarly. However, in electrical power systems, there are complexities that must be addressed in order for the system to be modeled in terms of structure and behavior. Two such complexities are:

- A component's behavior is a complex function of a number of parameters (voltage, current, impedance) operating simultanously; and
- The behavior of a component is properly described not merely by its input/output characteristics, but also by the characteristics of the devices both upstream and downstream from it.

The first problem was solved by the IPC developed by applying reasonable simplifying assumptions which affected the granularity of the model (i.e., the resulting model abstracted from certain properties that were deemed unnecessary) without hindering the ability to perform its required FDIR functions. Specifically, voltage was assumed to be constant everywhere in the system, and therefore could be modeled as a constant value, rather than something that needed to be computed. Secondly, impedances on the components were deemed too insignificant to contribute to the behavior of the system; they were also not modeled. The result was a model in which only current was propagated. Finally, loads were assumed to carry constant resistance, and therefore was also modeled as a non-computed value. This works for most kinds of loads; an exception is a fan, whose requirements for current changes somewhat over time (e.g., requiring an initial surge).

The second problem, modeling global effects of component behavior required more serious changes to the modeling process. One of the limitations of structure and behavior models is that the dual concepts of inputs and outputs imply a strong sense of directionality in behavior. This is not practical for certain kinds of changes to devices such as changes that result when an RPC opens or closes. Many researchers in model-based reasoning have been driven by this limitation of structure and behavior knowledge to seek a solution by representing more abstract forms of knowledge such

as functional knowledge (e.g., [15]). We selected a simpler strategy which did not require multiple models.

The specific problem we needed to address in building the model is the modifications to the equivalent resistance of the network caused by the opening or closing of RPC's. This resistance must be recomputed in order for the model to predict new current values at the sensors and elsewhere. The solution required the introduction of what we termed *meta-objects.* Meta-objects are used to represent non-directional behaviors in unidirectional models. These dummy components represent the global parameters in the system and are represented by mathematical equations that describe the relationships between these parameters. For example, they represent the equations needed to correctly recalculate the equivalent impedance of a circuit modified through tbe opening or closing of an RPC. Thus, when global phenomena occurring within the system limits the diagnostic abilities imparted by the unidirectional model, meta- objects are employed.

Consequently, from the standpoint of the predictor, behavior can be classified as local or global. Local phenomena can be handled by the standard input-output first-order transfer functions commonly used in unidirectional models. Global phenomena, on the other hand, must be represented by deriving an output from a number of input sources that are not directly (structurally) connected to each other either upstream or downstream. Due to the multi-directional nature of meta-objects, the use of meta-objects within the processing algorithms of a diagnoser must be restricted so that continuous looping is avoided.

Although pursued independently, the meta-component enhancement to the IPC represention apparatus seems similar to that explored recently by the developers of KATE [34]. Our opinion currently is that the concept of meta-objects represents only a partial solution of modeling global behavior, since their implementation still requires significant amounts of hard coded information about the system. Additionally, the IPC currently has no way of representing variable loads, such as motors, whose loadings may normally change throughout an interval of time. (This point is developed further below).

As noted, research on model-based diagnosis has focused mainly on representing structure and behavior knowledge, but more recently attention is shifting in order to consider representing other kinds of knowledge in a model. One reason for this shift is the need to find more efficient ways of controlling the reasoning process in the effort of diagnosing more complex systems. Power system behavior is often noted as an example of a physical system with complex behavior (e.g. [37]; [31]). As we'll observe below, the investigation described here incorporates some of the advances proposed by recent researchers.

3.2.2 Model-based Reasoning for Power Distribution FDIR

Of systems that employ a normal structure and behavior model for power system FDIR, Marple [12] provides an exemplary instance. Marple employs constraint suspension as its diagnostic strategy. It handles the complexity in propagating values in analog system models by using tolerances. It has been applied to actual power system hardware, and has an 85 per cent accuracy rate in identifying failures to components, including sensors.

The Marple effort is illustrative of the perception on the part of system developers using the model-based paradigm that enhancements to the paradigm are needed for diagnosing more complex devices such as power systems. This perception can be traced to observations made by the original developers of the paradigm (e.g., [5]).

We will henceforth refer to the time to action, or TTA, as the time it takes from the onset of a discrepency to the onset of the recovery process. TTA measures the speed of the diagnosis. As noted at the outset, one of our primary goals was to achieve a TTA which makes the IPC suitable for real time, on-board, use. It was decided, for this reason, to find an alternative to the constraint-suspension technique for diagnosis. The solution, roughly, involves replacing constraint suspension with additional knowledge about the connectivity of the system, as well as global assumptions about the behavior of the PMAD. Applying this alternative seems to have improved the efficiency of the diagnoser (although quantitative comparisions with the constraint suspension approach were not performed).

Finally, to adequately control power systems from massive failure, an automated

system must implement the ability to isolate components from the rest of the system. For example, when a fault to ground or to another conductor causes a quantity to be abnormal, the current will be greater than under normal conditions and the voltage will decay. If a voltage model is being used, the short circuit must be isolated so that the voltage can regain its normal level. Additional sources should not be made available because they would only serve to aggravate the problem by pumping more current into the short circuit.

4 Utilization of KATE

The diagnostic and control engine is the heart of the IPC prototype. It evolved from the Knowledge-based Autonomous Test Engineer (KATE), a shell developed for building model-based diagnostic applications by researchers at NASA Kennedy Space Center [19]. KATE consists of tools for building structure and behavior models to which a diagnostic reasoning engine can be applied. The reasoner employs a constraint suspension strategy. A predictor simulates the normal behavior of the target system which is compared to the actual performance of the target system itself. The diagnoser collects a set of suspects by placing in a list all components physically upstream from the discrepancy. Each suspect is individually tested for consistency by failing it purposely and suspending all of its constraints. The change is propagated throughout the system by using the behavioral knowledge of each component as well as a function which "inverts" the input-output function of each device to set values at the input of the device based on its output value. KATE also uses inversion to determine the action to undertake to isolate the failure and to establish an alternate source of power. KATE uses objects to represent its knowledge base, where a set of slots will contain the name(s) of the other components connected to its input and output.

The IPC inherited from KATE

1. The object-based model representation, including all the attributes for representing physical structure and behavior as well as the conceptual (ISA) hierarchy;

2. Some of the constraint suspension-oriented reasoning procedures for applying the model for control;

3. The basic algorithmic monitor-diagnose-control loop; and

4. Many of the low-level functions and procedures.

The IPC prototype differs from KATE, however, in crucial ways. First, the IPC is a translation of (the PC version of) KATE into C++. Second, as noted, the IPC replaces the constraint suspension approach to gathering and testing suspects with an approach using a combination of structural information and assumptions about the behavior of components. Second, the IPC implements the meta-object technique, described above, for modeling global behavior. As noted earlier, these components represent the global parameters in the system and are described by mathematical equations that describe the relationships between these parameters. For example, they represent the KCL equations needed to adequately describe the system. However, the concept of meta-objects only represents a limited means of representing global behavior because they include significant amounts of hard-coded information about the system that is difficult to implement as well as represent.

Figure 1 shows an example of the use of the representation of meta-objects in a model. Although the frames in Figure 1 (which actually represents part of the model of the PMAD) looks very LISP-like, it is parsed by the IPC into C++ objects. The definition of component RPC-Pl uses meta-object META-LCl-PORT to determine the equivalent resistance in a part of the circuit. Component RBI-P, located higher in the electrical hierarchy of the system, uses meta-object META-RBI-P which in turn uses META-LCl-PORT, in predicting the equivalent resistance in a larger part of the circuit which encompasses RPC-Pl. This avoids incurring the cost of using a meta-object if META-RBI-P used RPC- Pl instead of META-LCl-PORT. In such a case, the value of RPC-Pl must be calculated before META-RBI-P could use it. As can be inferred from Figure 1, a problem with meta-objects is that they demand significant computation, calculating the equivalent impedances every time the model is run, whether any changes in the structure of the system have taken place or not. This is especially costly when more than one meta-object is used in combination as in

```
 (deframe RBI-P
  (nomenclature "remote bus isolator (port side)")
  (aio rbi)
  (unit "amps")
  (source-path (and  (cstatus  gc-command-rbi-p)
                          (a//d-cstatus power-p)))
  (source power-p)
  (status (* 120 (meta-component meta-rbi-p)))
  (in-path-of current-rbi-p bus-p))

  (deframe META-RBI-P
  (nomenclature "Meta-component for port rbi")
   (aio meta-component)
   (source-path (cstatus gc-command-rbi-p))
(status
 (+ (if (cstatus gc-command-p1) (meta-component meta-lc1-port) 0.0)
 (+ (if (cstatus gc-command-p2) (meta-component meta-lc2-port) 0.0)
 (if (cstatus gc-command-p3) (meta-component meta-lc3-port) 0.0))))
(source t))

(deframe RPC-P1
 (nomenclature "remote power controller")
 (aio rpc)
 (units "amps")
 (source-path (and (cstatus gc-command-p1) (a//d-cstatus bus-p)))
 (source t)
 (status (* 120.0 (meta-component meta-lc1-port)))
 (in-path-pf bus-p1 current rpc-p1))

(deframe META-LC1-PORT
 (nomenclature "Meta-component for LC1 port side")
 (aio meta-component)
 (source-path (cstatus gc-command-p1))
 (status
    (+ (if (cstatus gc-command-p102) (/ 1 20.0) 0.0)
    (+ (if (cstatus gc-command-p103) (/ 1 60.0) 0.0)
    (+ (if (cstatus gc-command-p104) (/ 1 15.0).0.0)
    (+ (if (cstatus gc-command-p105) (/  1 30.0) 0.0)))))
 (source t))
```

Figure 1: Meta-Object Definitions

the above example. Therefore, they should be used sparingly and only within smaller domains. Another problem is that the impedances downstream from the meta-objects are hard-coded into the meta-object definition. This can introduce problems when loads are modified, either dynamically (i.e., a motor has a normal increase in torque demand when a compressor kicks in), or physically when one load is replaced by another of a different rating.

A final modification of KATE incorporated into the IPC was the ability to isolate components from the remainder of the system. KATE has traditionally been applied to process control systems, whose behavior does not require the ability of the controller to isolate components; hence, the needed change for the IPC.

In summary, the changes from the original KATE reflect the specialization of the IPC to use in electric power systems, as opposed to KATE's more general nature, which allows it to be used in process control systems and in digital electronics.

5 Utilization of the SSM/PMAD

The IPC developers needed to test the system on a near-real time environment to ensure the legitimacy of their research hypothesis. Consequently, they chose the SSM/PMAD as their testbed. The PMAD is a direct current power distribution system breadboard representative of a space station power system. It is interfaced to computer hardware for the purpose of testing power system scheduling, diagnosis and control software systems. The breadboard is supplied by two independent dc sources, the starboard supply and the port supply. Each of these are connected to a distribution bus through remote bus isolators (RBI's). RBI's are switches used for isolation of the power supplies and are not capable of current interruption. Each of the two busses supply the power to three load centers, Load Centers 1, 2, and 3, through 3 kW remote power controllers (3k RPC's). The RPC's are solid state dc circuit breakers capable of interrupting fault currents. Figure 2 shows a schematic representation of the PMAD distribution network. Each load center has a number of loads being supplied by its bus. These loads consist of lights, fans, or simply resistors, and are supplied from their respective load center through 1 kW RPC's.

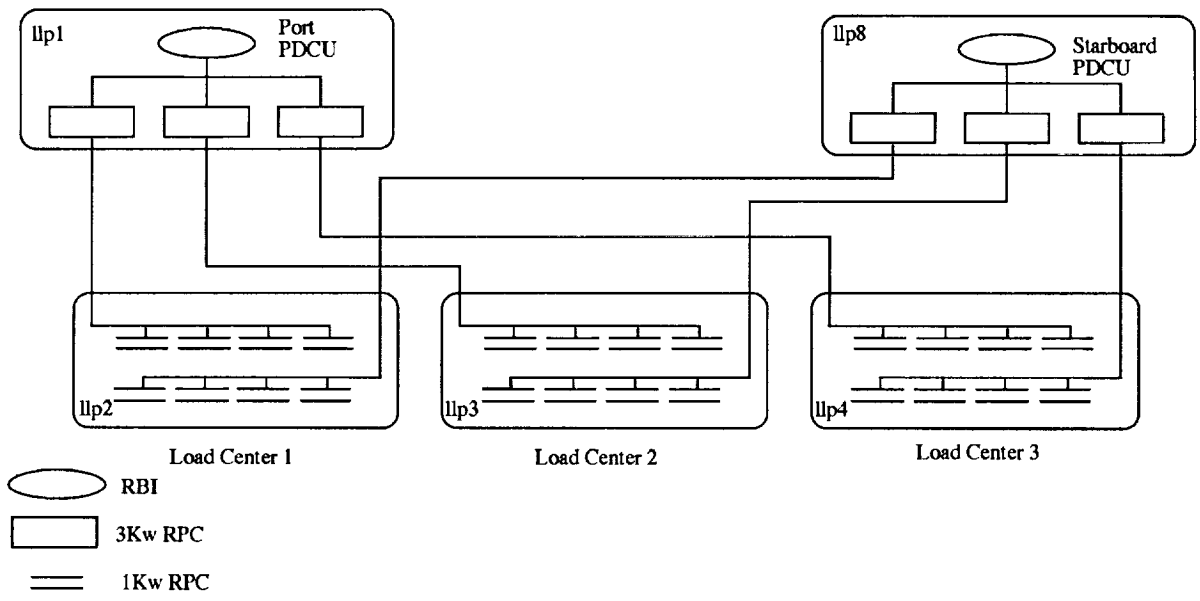

Figure 2: The SSM/PMAD Breadboard Schematic

Each load center has two busses, one coming from the starboard supply and the other from the port supply. A set of critical loads such as cabin air supply and emergency lights are fed from both supplies through two redundant paths, so that if one supply or bus is lost, they can be alternatively fed from the other. The IPC prototype testing centered around the detection of systems faults, the isolation of the faulted component in the system, and the establishment of the redundant path to these critical loads. Since the redundant loads were contained in Load Center 3, the bulk of the testing was carried out in Load Center 3. Sensing devices consist of current sensors and voltage sensors. Each RPC contains a current sensor that measures the current flowing through it. Additionally, external current sensors at various locations such as the RBI also measure current in the circuit. Bus voltages are measured, but the signal received is a discrete value when the voltage reaches a specific percentage of full voltage. Thus, they were not used, and the current model of the PMAD was implemented in the IPC prototype.

The RBI's and the RPC's are controlled through a set of networked Lowest Level Processors (LLPs), which form the interface between the controlling computer and the switchgear. The LLP's serve to read the sensors as well as to carry out any action

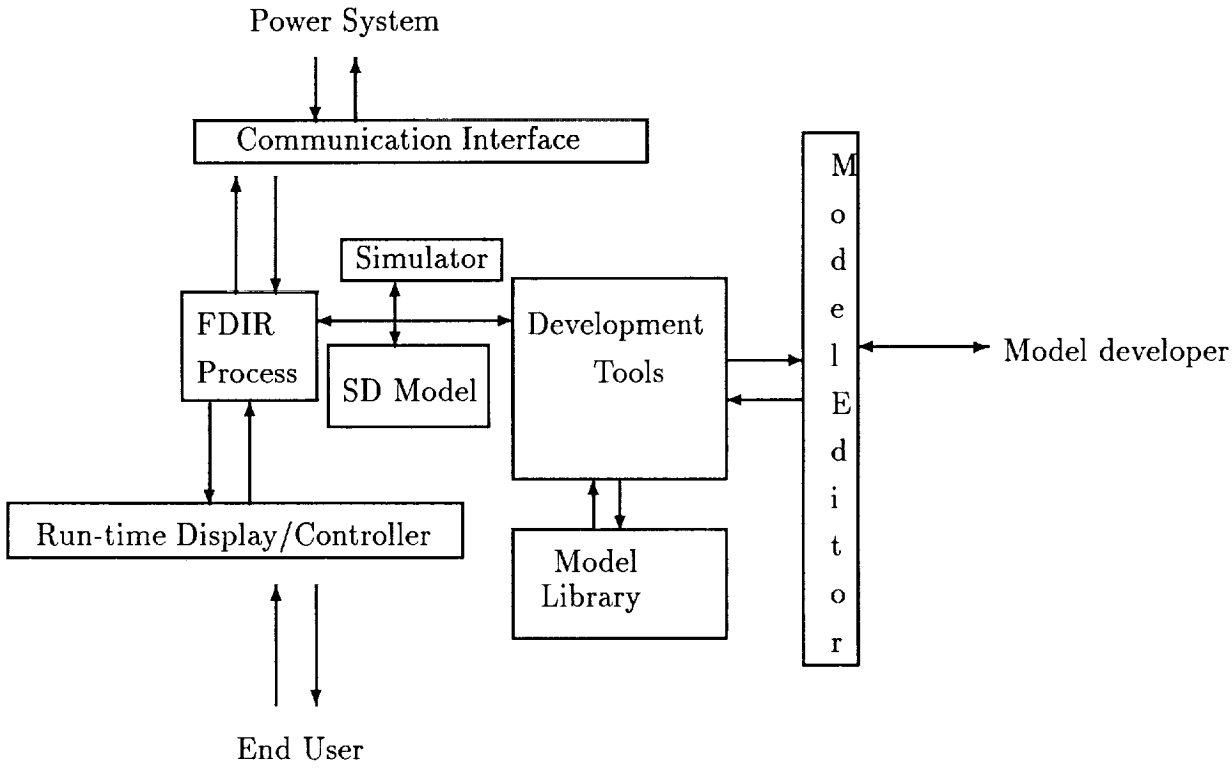

Figure 3: Intelligent Power Controller Architecture

commands, such as opening or closing switches, directed to the RPC's and RBI's. The IPC prototype interfaced directly with these LLP's. Since the LLP's were networked to the Internet system, tests of the IPC could be done remotely from the primary research sites of Orlando and Melbourne, FL.

6 IPC Architecture

This section summarizes the technical discussion by presenting a description of the architecture of the IPC. The Intelligent Power Controller (IPC) consists of two main subsystem, data bases and libraries. The overall architecture is displayed graphically in Figure 3. The two large subsystems identifiable in the figure are: the real time

power controller (IPC/RT) and the graphical modeling tool system (IPC/GMT). The RT consists of the hardware interface, the simulation tool, the FDIR system, and the real time controller and display. The IPC/GMT consists of a model library, a model development tool, a simulation tool (which it shares with the run time subsystem), and an icon editor. The end user builds and tests models of a target system through interactions with the icon editor. Because the IPC/GMT subsystem was not a requirement of this contract, and the contract award was not applied to the development of this subsystem, this document will not contain a discussion of the IPC/GMT. Rather, in the following subsections, we discuss in greater depth the run time design of the IPC/RT, followed by sample run time sessions.

6.1 The IPC/RT System

The IPC/RT is driven primarily by user interaction. Its run time environment can be divided into two modes of operation. The first mode provides the CAD facilities that are needed to construct a useful display of a model. The second mode of operation provides a runtime environment that allows the user to easily interact with the IPC, and to visually monitor the system.

When the IPC/RT is first executed, the user must specify what model files are to be loaded. Initially, the screen is grey, with only two menu options available under the button **File** on the main menu bar. The user can either **load a model**, or **exit** the IPC/RT. Once the user loads a model, the system follows the actions shown in Figure 4. This figure shows how the parser begins the process of translating the model information into the iconic display. Once the model has been parsed, the IPC/RT displays the objects on the screen, along with any relevant connections.

The actions the user may perform in the two operating modes are implicitly mutually exclusive. On the runtime side the user should not be able to modify the layout of the screen. In this mode, the user is only interested in issuing commands to the IPC and viewing the results. On the CAD side, the user should not be able to issue any runtime commands, but should be able to modify the screen as well as the model. There is no impact of screen changes during a runtime session.

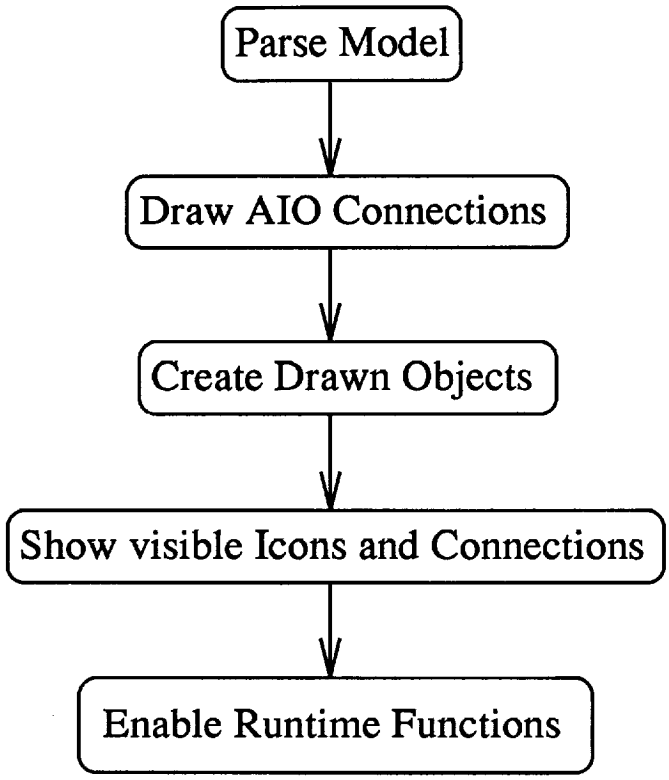

Figure 4: Program Flow for Load Model

6.1.1 CAD Facilities Overview

Once a model is displayed on the screen, the IPC/RT enables many of the other facilities for the user. Initially, the IPC/RT is in the CAD mode of operation. The Xview Notifier, a process for retrieving data from file descriptors, handles all system activities, including user interactions, and communicating with other systems. Figure 5 shows the role of the Notifier.

The CAD environment implements the following facilities:

- The CAD mode allows the user to customize the arrangement of icons on the display. The user selects an object to be repositioned with the left mouse button, and while holding the button down, drags the icon to the desired position. The connections associated with the icon will automatically be updated when the user releases the left mouse button.

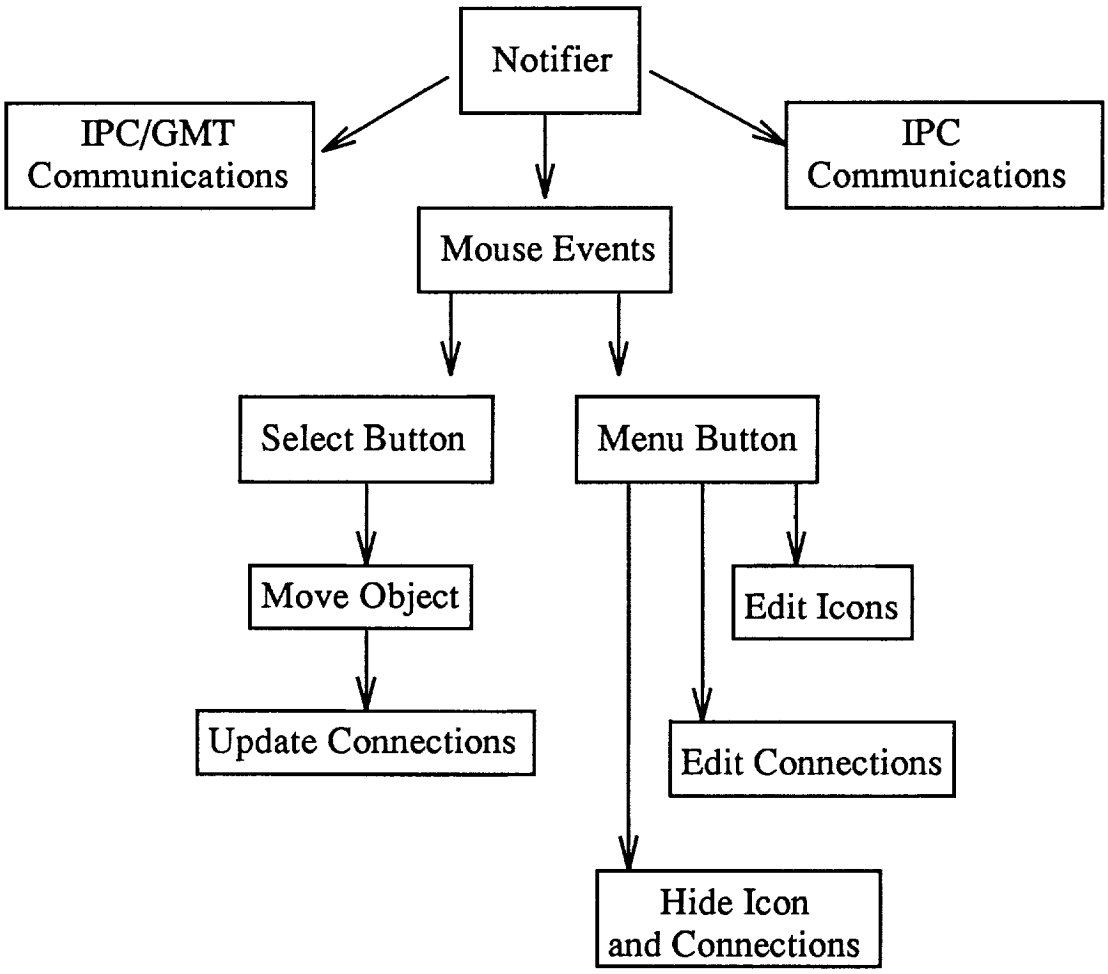

Figure 5: IPC/RT CAD Facilities

- The user is also able to edit the icons to be used by an object. Multiple icon pixmaps can be specified, along with attributes specific to each of them. These icon attributes are edited by the user through a dialog box.
- The CAD mode of operation also allows the user to edit the attributes of each connection associated with an object. The attributes of each connection are edited by the user through a dialog box.
- Lastly, while in the CAD mode, it is possible to have another support tool contending for write permissions on the model data files. The Notifier warns the user of any contentions with other systems. The IPC/RT is considered a server in terms of communications, and any other support tools are clients. These clients may at times need to modify the model files, and precautions are taken to prevent losing edits made by the user in the IPC/RT CAD mode.

The editing functions in the CAD mode are accessable through a set of menu buttons, starting with the top-level *mode* button, which allows the user to enter the editing mode, and progressing through the set of editing buttons corresponding to the functions just listed.

6.1.2 Runtime Facilities Overview

The runtime facilities allow the user to easily control and monitor the IPC. Once a runtime session is initiated through the main menu bar, the Notifier serves many different roles. Figure 6 depicts the activities of the Notifier.

The runtime environment implements the following facilities:

- Four text windows are provided for monitoring the output of the IPC. They comprise the following:
  1. A *warning window* which receives messages that are intended to attract the attention of the IPC developer. Warnings indicate events such as a fault being detected in the system;
  2. A *recovery window* which receives the names of the objects that are recovered after the FDIR process is completed;

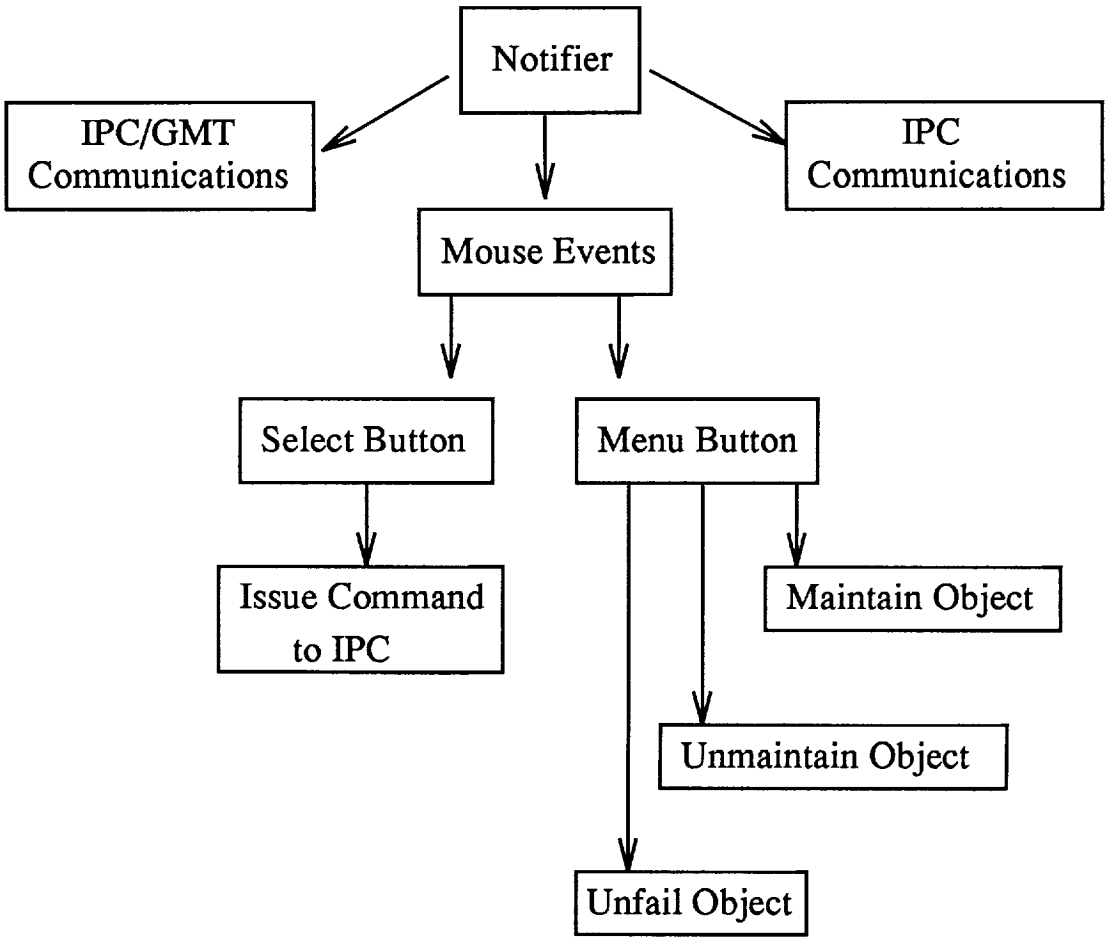

Figure 6: IPC/RT Runtime Facilities Overview

3. A *general window* which receives messages that are important to the IPC developer, but cannot be classified as a warning; and
4. A *main window* which displays the raw output of the IPC.

- The user is able to use the mouse to interact with the IPC/RT. The left button of the mouse issues control commands to the IPC. If the icon being clicked on is a control function, called a *GC-COMMAND*, the Notifier will send a message to the IPC to toggle the state of the object;
- The right mouse button, which is the menu button, also gives the user a different menu than in CAD mode. This menu has the following three options:
  1. **Maintain**. This menu option allows the user to tell the IPC to maintain an object at a specified value. Once an object is on the IPC's maintain list, the IPC will attempt to restore power through a functioning redundant path. An object that is being maintained is highlighted with a green border.
  2. **Unmaintain**. This menu option removes an object from the IPC's maintain list.
  3. **Unfail**. This menu option allows the user to tell the IPC to remove an object from its list of failed objects. The IPC is then able to determine when previously failed hardware has been corrected. It can then use the component as a possible path for restoring power to the critical loads.
- The IPC/RT updates the current sensor values whenever the "display measurements" command is processed by the IPC. The value is written over the icon that represents a current sensor. The user can also type "d m" into the command window, or press the **UPDATE SENSOR VALUES** button on the main menu bar. If the IPC is in the FDIR process, there may be some delay before receiving the new measurements.
- The IPC/RT also updates the current state of the switches. The IPC/RT also writes the letter that represents the state onto the left hand side of the icon.

The letters 'N','F','T', and 'U' denote the switch state as being on, off, tripped, and unusable respectively.

- A command window is provided to allow the user to directly issue commands to the IPC in a text format, and is primarily for the use of the IPC developers.
- The IPC/RT utilizes color highlighting to clearly show the objects that are involved in the FDIR process. All suspects are given a yellow border, and any failed objects have a red border.
- The Notifier accepts clients that need to communicate with the IPC/RT, such as the IPC/GMT. Once a client is established, all communications with the client and the resulting warnings to the user that may occur, are issued by another parsing routine invoked by Notifier.

By providing these capabilities, the runtime environment is able to provide a fast and robust method of interaction with the IPC.

6.1.3 Iconic Representation of PMAD Objects

The PMAD images that are supplied to the user were drawn using a public domain color image drawing and editing tool call **pixmap**. The graphics format that was used in this project is the **XPM** format. This common format allows the user to utilize any drawing package desired, even an image scanner, as long as the final output is converted into **XPM**. This conversion can be done with the **Pbm Plus** package that is available on Internet.

An object instance that has no previous icon definitions is given a default image which the user can interactively change. For example, an RPC is displayed in three parts: the object itself, a measurement, and a command. The supplied interface has three images that are positioned next to each other, to give the appear of a single object, the **RPC**. Figure 7 shows the three icons that make up the RPC. Future enhancements would allow the user to specify a single icon for multiple objects, a group, or to simply group several icons together for easier repositioning.

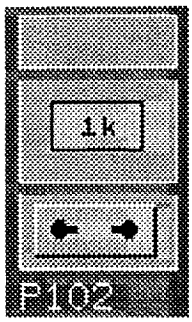

Figure 7: Example Iconic RPC

6.1.4 Frame Data Validation

In the process of parsing the files that define the model, the system is capable of validating each frame definition. Validation takes the form of a recursive descent parser. The validation currently performed is only for the syntactic aspects of the model.

Another step toward the validation of the model lies in the testing for cycles within the model. This task is relatively trivial due to the design of the IPC/RT, and the traversal methods have already been defined within the classes.

6.2 Sample Runs

To allow the reader a sense of the IPC user's point of view, this section summarizes the runtime session which occurred during the demonstration of the IPC at NASA Marshall Space Flight Center as part of the requirements of this contract. During the demonstration, the IPC performed perfectly, and the IPC/RT was able to provide a fast and easy runtime environment. A more detailed analysis of the tests will be provided in the next section.

The demonstration consisted of a set of six tests which served to prove the capabilities of the IPC. An additional scenario was accidentally introduced by an inexperienced user of the IPC/RT, and is also recorded in the playback. The IPC performed as expected, even though the accidental case had not been attempted previously. The fact that the user was able to introduce a new scenario shows how easily the IPC can

be manipulated through the use of the IPC/RT.

The following sections present snapshots of three of the scenarios performed during the demonstration. The color highlighting of the objects in the snapshots indicate the following conditions.

- **Green** An object with a green border is a critical load that is to be maintained by the IPC. If its current power source fails, the IPC needs to take action to restore power through a redundant power source.
- **Yellow** An object with a yellow border indicates that it is a suspect for the fault(s) detected in the system.
- **Red** An object with a red border indicates that it is a failed (unusable) component and the cause for a fault in the system.

In each of the following scenarios, the critical loads PRPC-30620 (a fan) and PRPC-30418 (a bank of lights), are being maintained by the IPC. Normally the IPC is monitoring the system as shown in Figure 8.

6.2.1 Fault with Recovery

In this scenario, a hard fault is placed on Remote Power Controller (RPC) P306, which is currently the power source for the critical load PRPC-30620. The IPC begins the FDIR process and indicates the suspects for the fault. Figure 9 shows the IPC/RT screen. On a color terminal, the suspects are highlighted in yellow.

Once the suspects have been determined, the IPC isolates the cause of the fault. The IPC fails RPC-P306, and restores power to PRPC-30620 via RPC-S320. Figure 10 shows the failed component and the new source of power for the critical load.

6.2.2 Fault with Double Recovery

In this scenario, a hard fault is placed on Remote Power Controller (RPC) P3, which is currently the power source for RPC-P304, RPC-P306, and RPC-P307. RPC-P306

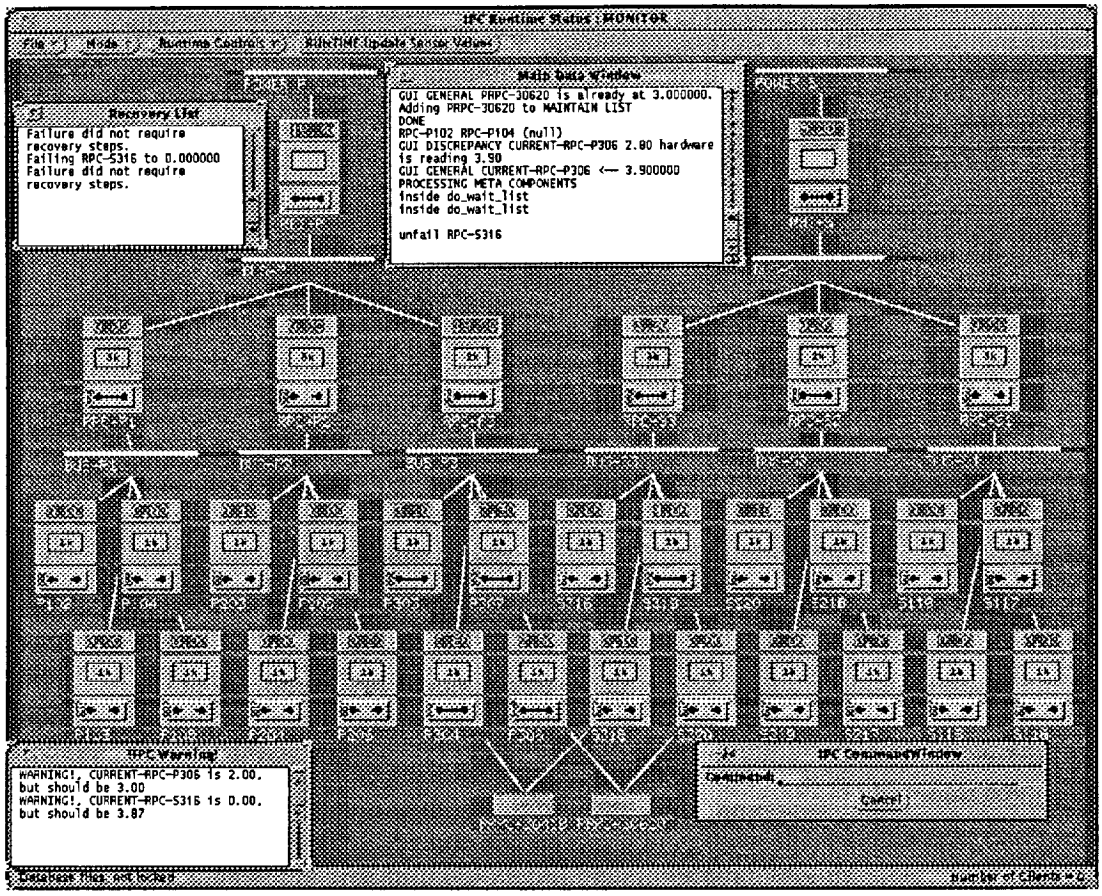

Figure 8: Monitoring the SSM/PMAD

is the power source for the critical load PRPC-30620, and RPC-P304 is the source for the critical load PRPC-30418.

The IPC begins the FDIR process and indicates the suspects for the fault. Because RPC-P3 supply power to RPC-P304, RPC-P306, and RPC-P307, they each trip on under-voltage. This results in both critical loads losing power. Figure 11 shows the IPC/RT screen (again, in color, the suspects are highlighted in yellow). Once the suspects have been determined, the IPC isolates the cause of the fault. The IPC fails RPC-P3, and restores power to PRPC-30620 via RPC-S320, and also restores power to PRPC-30418 via RPC-S318. Figure 12 shows the failed component and the new sources of power for the critical loads.

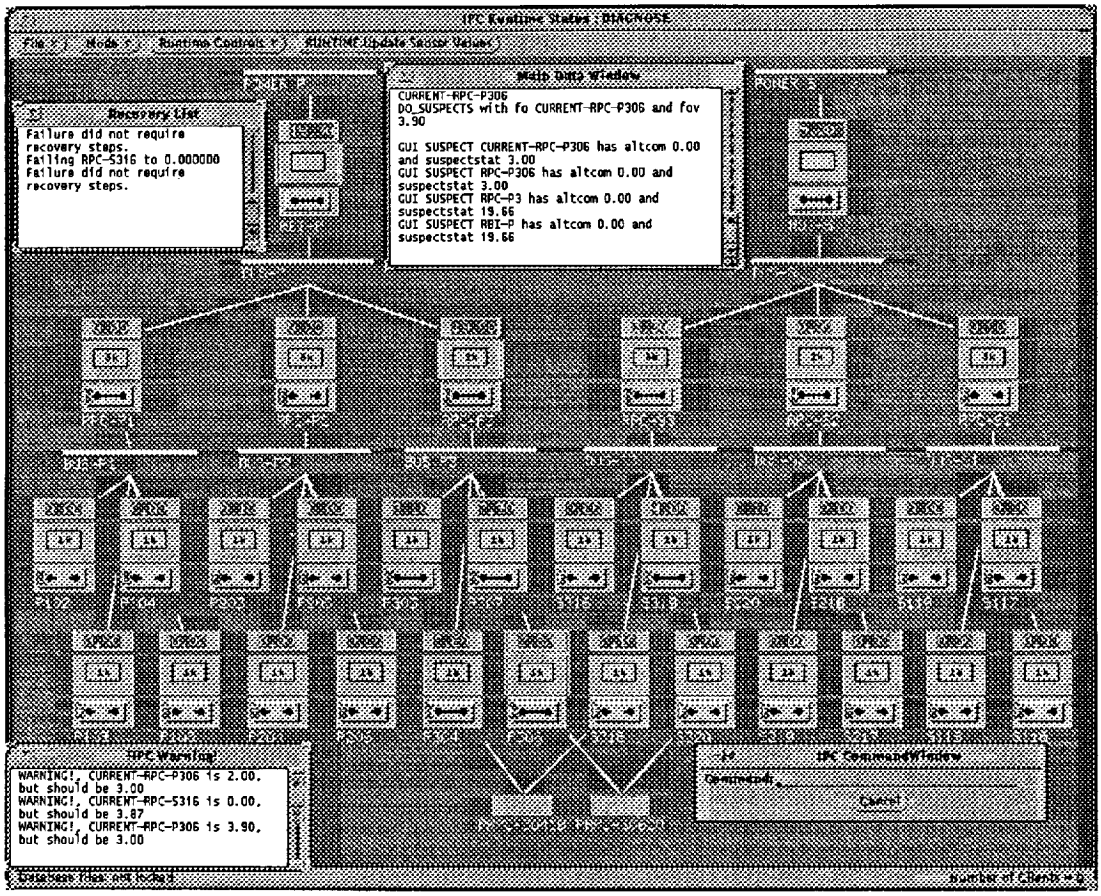

Figure 9: Scenario 1: Fault to RPC P306

6.2.3 Multiple Fault without Recovery

In this scenario, hard faults are placed on two RPCs, P303 and P307. These RPC's are not supplying power to any of the critical loads. The IPC begins the FDIR process and indicates the suspects for the fault. Figure 13 shows the IPC/RT screen at this point. Once the suspects have been determined, the IPC isolates the causes of the faults, and the IPC fails RPC-P303 and RPC-P307. Figure 14 shows the failed components.

The screen shots presented in this section show the SSM/PMAD model display, and how it changes during a real runtime session. The playback data used to get these snapshots was taken from the demonstration given at NASA MSFC. During

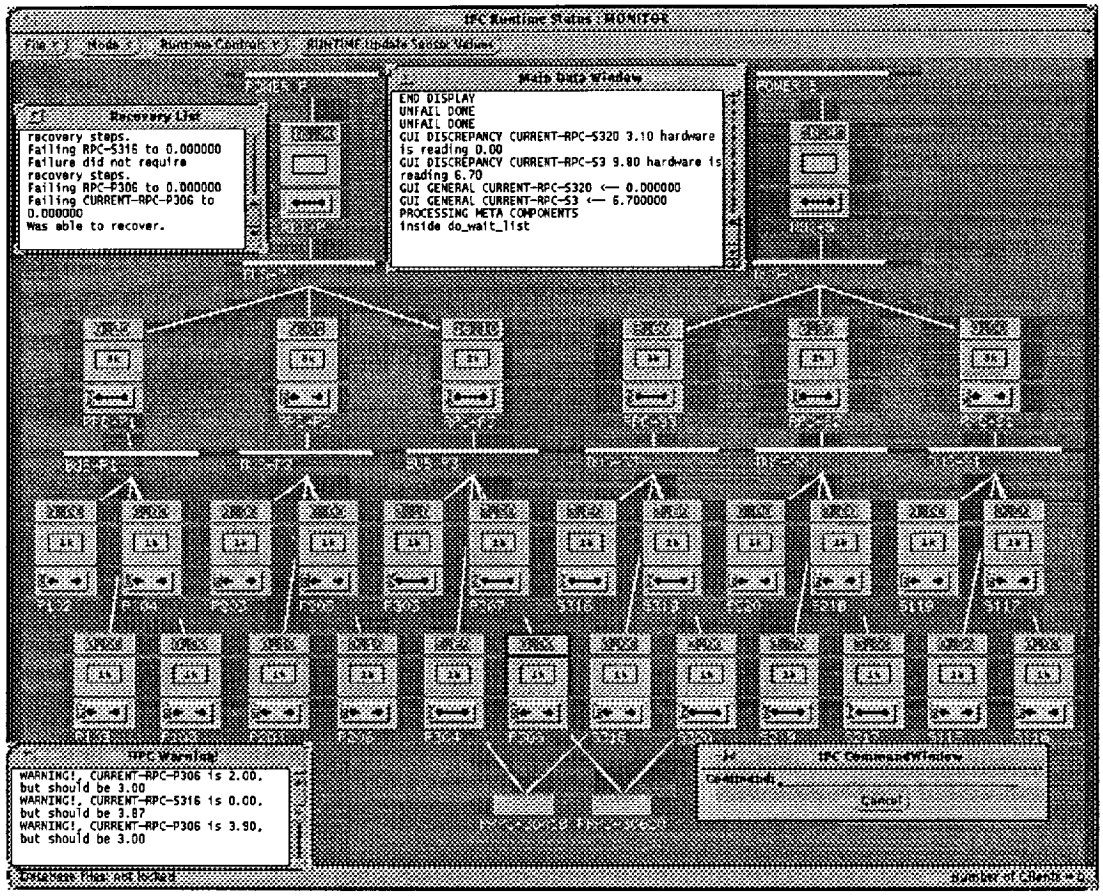

Figure 10: Scenario 1: Failure of RPC-P306 and Power Restored to PRPC-30620

this demonstration, the IPC/RT clearly demonstrated its ability to meet its primary requirements of being fast and easy to use.

7 Testing the IPC

This section describes the testing undertaken to prove the effectiveness of the IPC in performing FDIR, using the PMAD testbed.

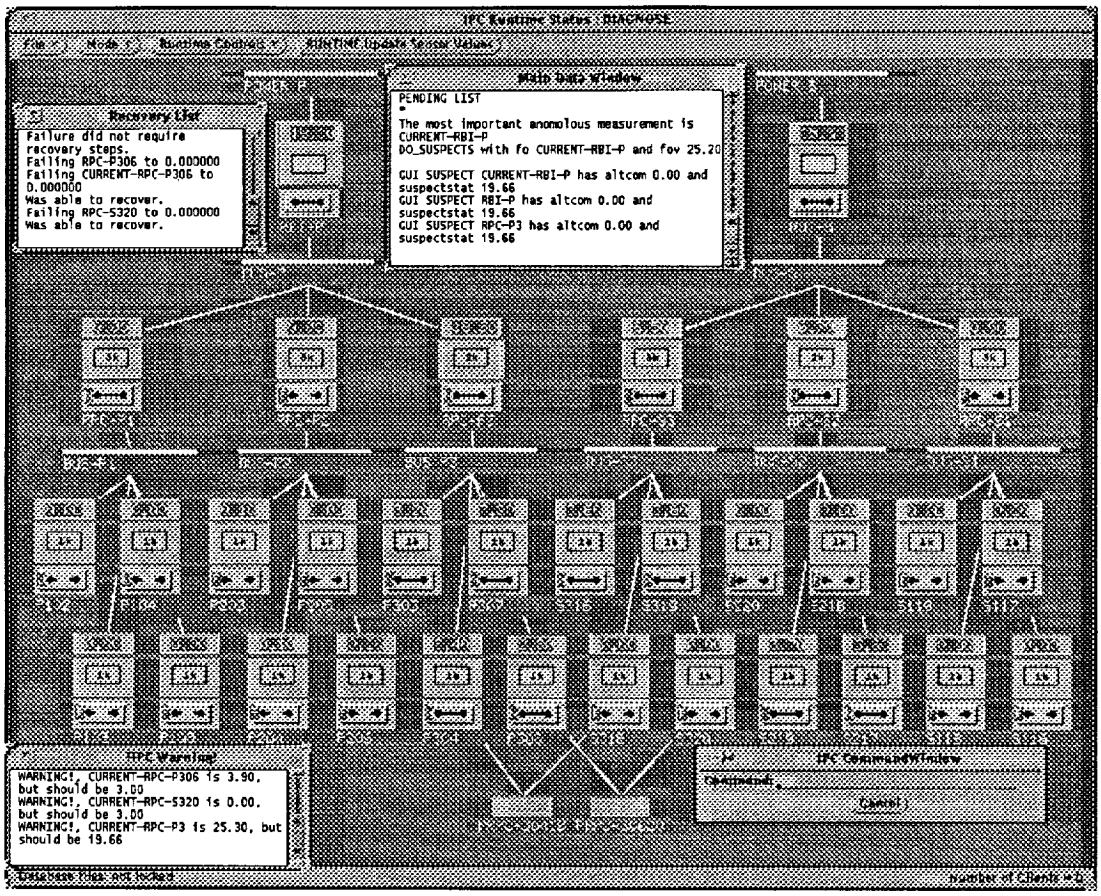

Figure 11: Scenario 2: Fault to RPC P3

7.1 Test Scenarios

There was a significant number of tests performed on the IPC, using the PMAD. Some of these included putting direct short circuits to ground at various locations in the system, such as on the load side of the 1k PRC, and on the load center side of the 3k Remote Power Controllers (RPC's). This type of fault was referred to as a hard fault, and it required the IPC prototype to recognize the discrepancy in the current sensor reading, find the "failed" component (usually an RPC), and if opening the RPC would cause loss of power to a critical load that had a redundant path, to establish an alternate path by closing the appropriate RPC. The RPC's in the PMAD have internal sensors that cause them to trip on hard faults; on accumulated overcurrent

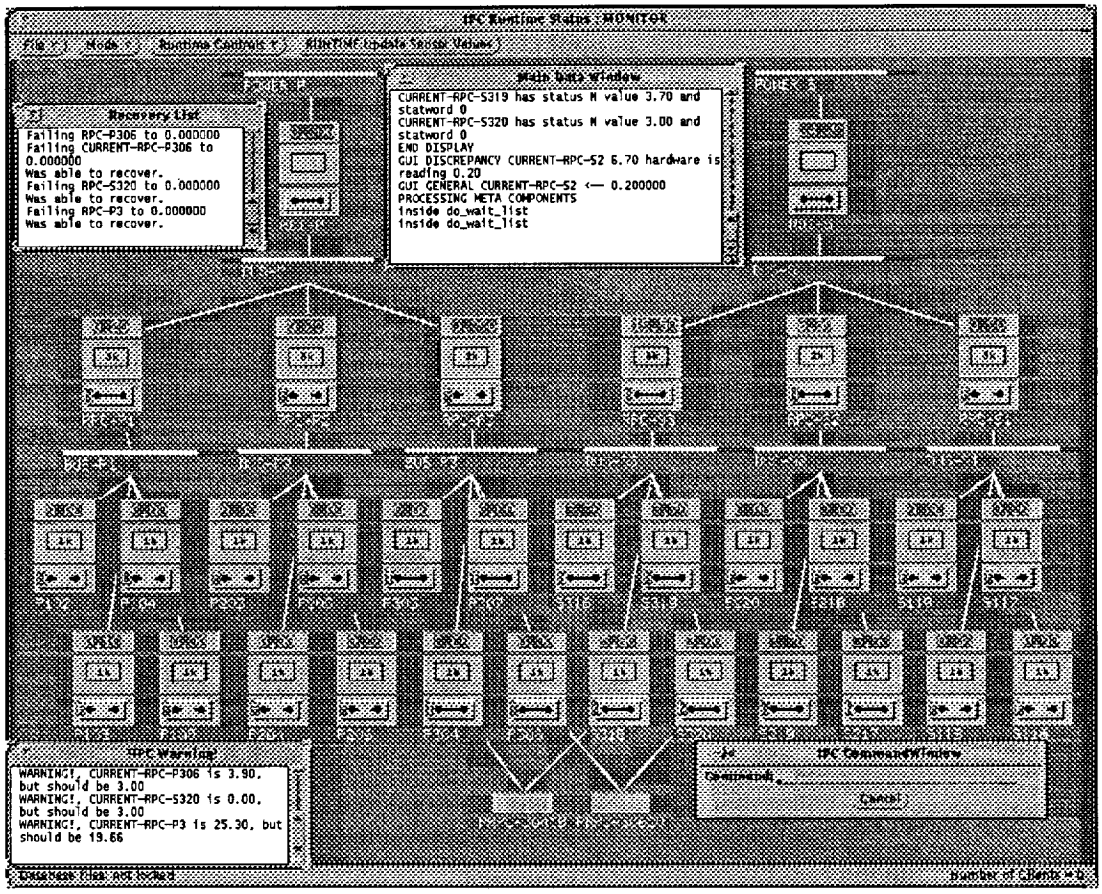

Figure 12: Scenario 2: Failure of RPC-P3 and Power Restored to Critical Loads

over time in cases of lower level faults (I^2t), or on undervoltage. The IPC prototype, therefore, would see the discrepancy not as a difference in current magnitudes, but rather, as the unexplained opening of a RPC by its internal mechanism. It would thus only have to establish redundant paths to critical loads.

Another type of test consisted of short circuits to ground through impedances (called soft faults). The current levels generated by these faults are significantly lower than that of hard faults, and are thus more insidious. They are, therefore, difficult for conventional protective devices to detect and protect against because the level of overcurrent may be below not only the instantaneous trip level of an RPC, but also below the I^2t pickup level where there would be no trip regardless of the time elapsed. Yet, soft faults can be quite destructive. Since the RPC's would not

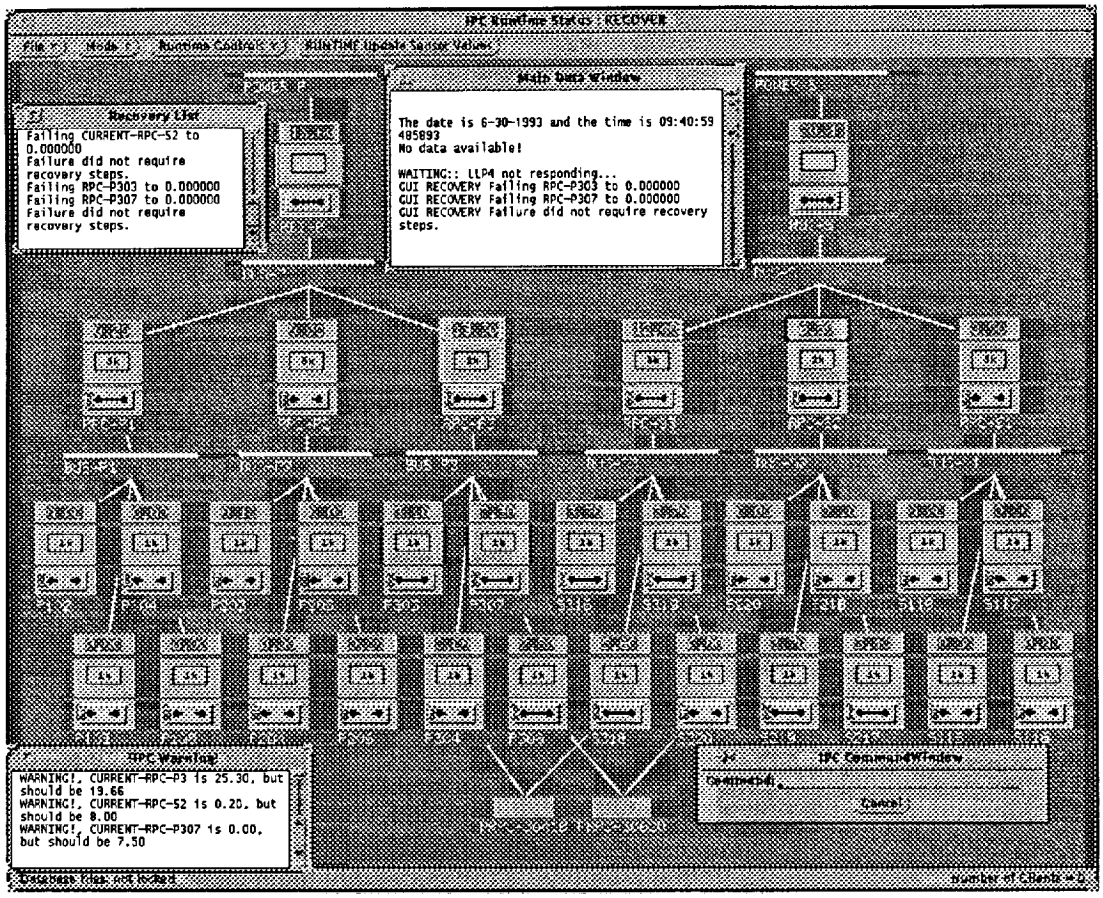

Figure 13: Scenario 3: Fault to RPCs P303 and P307

trip themselves automatically for soft faults, the IPC, in this case, is tasked with detecting and diagnosing the problem, isolating the fault location (usually the closest upstream RPC), and initiating recovery action, if such is warranted.

Sensor failures were also simulated by disconnecting the leads on current sensors being used by the IPC. The action expected of the IPC in this situation was to declare the sensor to be erroneous and label it as unusable, but allow the rest of the system to continue normal operation. This represents a difficult task for local relaying schemes, as well as for the state of the art in global monitoring systems which employ other artificial intelligence techniques.

Lastly, multiple (two) independent short circuits (hard and soft faults as well as combinations of both) were placed on the system simultaneously. The prototype was

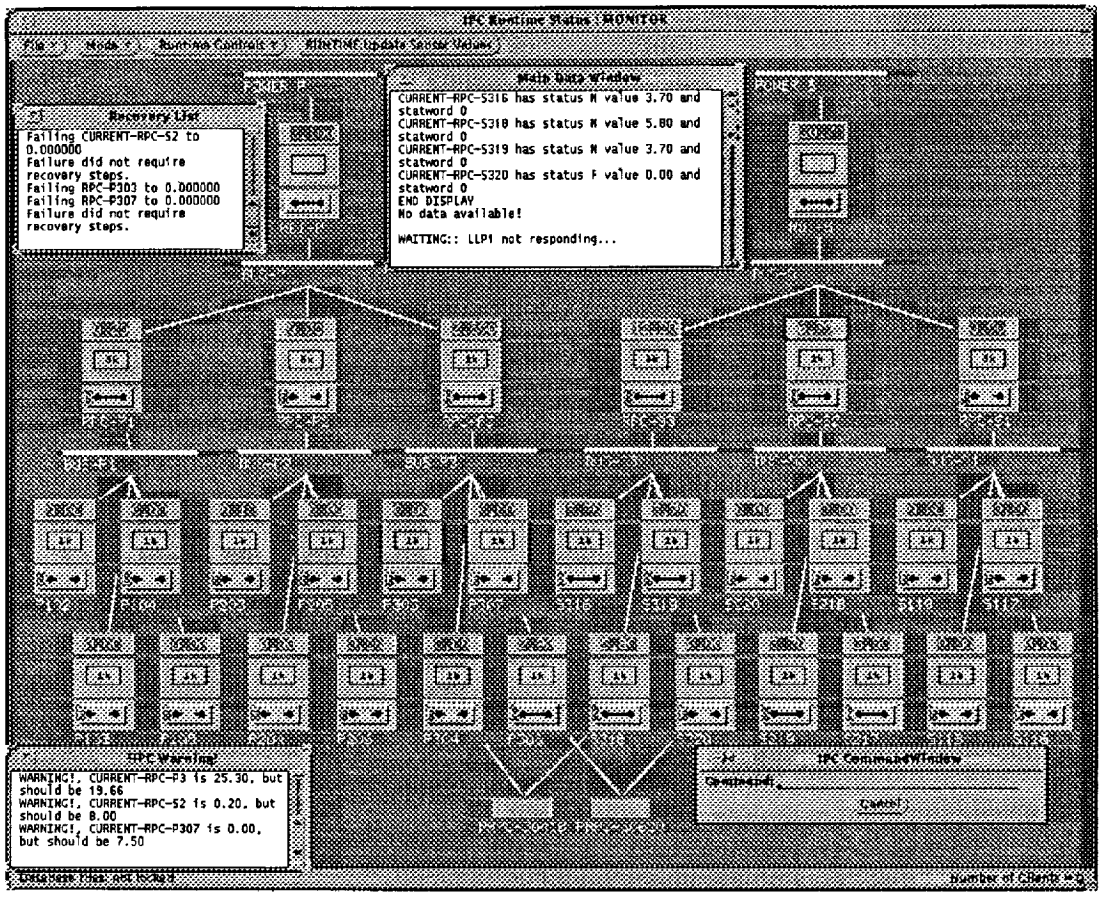

Figure 14: Scenario 3: Failure of RPC-P303 and RPC-P307

expected to identify all of the fault conditions, isolate them and initiate recovery action. This likewise represents a difficult case for local protection schemes as well as for global monitoring systems that use artificial intelligence techniques.

The hard and soft fault tests were all executed in Load Center 3 due to its inclusion of redundant loads. No such redundancy was present in the other Load Centers. Tests on failed sensors, however, were done in Load Center 2 because that is where the failable sensor was installed. This test consisted of the IPC monitoring the PMAD while under normal operation for a sufficiently long period of time to establish normal conditions in the system. Then, the leads to the failable sensor were removed, which caused it to read zero.

There were two objectives to the test program: a qualitative and a quantitative.

The qualitative objective was to determine whether in light of the sensor readings observed, the IPC reached the proper diagnosis and carried out the appropriate corrective action.

The quantitative objective of the tests was to measure the time-to-action of the test. The time-to-action, as noted earlier, is the time taken by the IPC to complete the isolation of the failed component, starting from the time the fault was placed. The results of this quantitative aspect of the test phase were important because, as noted earlier, in order to represent a usable tool in the real, earth-bound world, the IPC must have an adequate time-to-action. A suitable time-to-action of the IPC when fast local tripping capability is available to protect against hard faults (such as is the case with the PMAD) is in the order of a few seconds (10 to 15). In such circumstances, the IPC would serve as secondary protection against hard faults and primary protection against soft faults and sensor failures not normally detected by the local protection, as well as for the recovery of the system when affected by either hard or soft faults. If the IPC is to be used as a primary means of protection for hard faults, however, an adequate time-to-action would have to be in the order of 0.5 to 1.0 second. This is based on the fact that low voltage distribution breakers in earth-bound power distribution networks have fault clearing times of less than 10 cycles of 60 Hz current. This equates to 0.167 seconds. If additional time is allotted for data acquisition from sensors, then an upper bound of one half to one second for a time-to-action would be considered adequate.

The time-to-action, however, can be a misleading measure of the response time of the IPC itself, since this parameter includes any and all communications network delays in getting the sensor data from the LLP's to the IPC, as well as relaying the controlling command from the IPC to the LLP's. There were a total of 13 sets of tests designed for the evaluation phase of the investigation. Table 1 describes the general objective of each test. All but one were successfully executed numerous time (between 10 and 35 times).

Due to the overall symmetry of Load Center 3, and the general objectives of the tests, the same test was executed on different, but hierarchically and functionally identical, sets of components in the system. Table 2 describes the components most

Test Number	Test Description
1	Soft fault in 1K circuit with critical loads maintained
2	Hard fault in 1K circuit with redundant loads maintained
3	Soft fault in 3K circuit with redundant loads maintained
4	Hard fault in 3K circuit with redundant loads maintained
5	Soft fault in 1K circuit with no recovery
6	Hard fault in 1K circuit with no recovery
7	Soft fault in 3K circuit with no recovery
8	Hard fault in 3K circuit with no recovery
9	Sensor failure at 1K RPC
10	Sensor failure at 3K RPC
11	Multiple faults with no recovery
12	Multiple faults with two redundant loads maintained
13	Multiple faults with one redundant load maintained

Table 1: Test Objectives

Test Number	Failed Object	Isolated Object	Recovered Object
1	RPC-P306, CURRENT-RPC-P306	RPC-P306	S320
2	RPC-S320	RPC-S320	P306
3	RPC-P3	RPC-P3	S318, S320
4	RPC-S3	RPC-S3	P304, P306
5	RPC-P306	RPC-P306	NR
6	RPC-S316	RPC-S316	NR
7	RPC-P3	RPC-P3	NR
8	RPC-S3	RPC-S3	NR
9	NOT TESTED		
10	CURRENT-RPC-S2	NR	NR
11	RPC-P303, RPC-P307	RPC-P303, RPC-P307	NR
12	RPC-S318, RPC-S320	RPC-S318, RPC-S320	P304, P306
13	RPC-P303, RPC-P304	RPC-P303, RPC-P306	S318

Table 2: Example Runs for Fault Tests

frequently used to carry out each of the tests. The symbol 'NR' indicates 'not required'. This indicates those tests that did not require either component isolation, or recovery of redundant loads. Tests 11, 12, and 13 were tested with a combination of hard and soft faults.

7.2 Test Results

The IPC prototype successfully met its qualitative objectives under all test conditions executed. Test 9, however, required additional hardware to implement and was not executed. The tests showed that the IPC prototype was successful in diagnosing the problem, isolating the fault, and where required, carrying out appropriate control action. This control action consisted of supplying all redundant loads from alternate sources. This was true for all 12 cases executed. Furthermore, the IPC testing confirmed the ability of model-based reasoning to diagnose unexpected failures. As noted earlier, during one testing session, a casual user improperly opened one of the RPC's feeding a redundant critical load (the other RPC, meanwhile was already open). Since the load had been designated as critical, and thus for a certain level of current to be maintained to it, the IPC took it upon itself to immediately close the other RPC, thus re-establishing a power source to the critical load.

The quantitative objective of the test program was to measure the time-to-action exhibited by the IPC for some of the tests. Table 3 depicts the worst, the best, and the average times recorded for a series of tests. All times depicted are in seconds. The number of Tests column of Table 3 represents the total number of times the test was run successfully. However, timing measurements were not performed for all test runs. In fact, time-to-action data for tests 5 and 6 result from only one test run, and should be interpreted with this fact in mind.

It was found that the times-to-action ranged from slightly over 7 seconds to as much as nearly 34 seconds, with the average being about 10 to 12 seconds. These times-to-action are considered sufficiently fast for real time deployment to isolate hard faults in the presence of fast local tripping devices such as found in the PMAD with the RPC's. Such an arrangement prevents the flow of the high fault currents that typically result from hard faults. Soft faults, on the other hand, are generally

Test	*Number of Tests*	Worst Time		Best Time		Average Time	
		Isolate	*Recovery*	*Isolate*	*Recovery*	*Isolate*	*Recovery*
1	30	11.82	2.01	10.65	1.45	11.21	1.73
2	35	11.89	2.02	10.39	1.52	10.97	1.78
3	15	9.47	3.85	7.86	1.94	8.93	3.22
4	20	Time-to-action not measured					
5	30	9.4	NR	9.4	NR	9.4	NR
6	30	33.92	NR	33.92	NR	33.92	NR
7	15	Time-to-action not measured					
8	20	Time-to-action not measured					
9	0	Test not performed					
10	15	8.9	NR	7.4	NR	8.1	NR
11	20	14.73	NR	10.57	NR	12.80	NR
12	15	Time-to-action not measured					
13	10	Time-to-action not measured					

Table 3: Results of IPC Tests on PMAD Breadboard

not considered to be time critical due to their low current level, thus making those times-to-action acceptable. The same applies for sensor failures. Since soft faults and sensor failures are not typically detected by such local protection schemes, the presence of the IPC provides a significant advantage.

Nevertheless, these times-to-action are not sufficiently fast in the absence of fast local interrupting devices. Times-to-action of one second or less will be required in order to make IPC applicable to such duties.

Some of the excessive delay can be attributed to the use of the Internet to execute the tests remotely from Orlando to Melbourne. In order to quantify this delay, additional tests were done locally as well as remotely. Test 2 was selected for this experiment since it is considered to be one of the most common faults in actual power systems. The objective was to compare run times without any optimization to obtain a gross representation of the internet delay times. The results are depicted in Table 4. A comparison of the times showed approximately -1/2 to 18 seconds difference in time-to-action. The worst REMOTE time was obtained in the afternoon (EST) which is a heavy traffic period for Internet given the start of business hours on the West Coast.

This result shows that the effect of an Internet delay cannot be determined by general comparisons of run times. Also, it may show that Internet delays are also

Test	Remote		Local	
	Best	*Worst*	*With Delay*	*Without Delay*
2	34.97	53.14	35.54	22.16

Table 4: Speed Comparison of Local vs. Remote Testing

occurring significantly at Huntsville. It should also be noted that the test at NASA-MSFC was run on a Solbourne while the remote testing was done on a Sun Sparc 1+ which is somewhat faster. This may explain the slower time for LOCAL. All times were compared using the computer clock time elapsed from a discrepancy detection to a fault isolation and then to a recovery in hundredth's of a second.

A second, and more significant, objective of this comparison was to compare the running time to a version of the code that was exempt from internal Internet delay waits that were embedded in the interface code in order to run the IPC remotely. The other value displayed in Table 4 (Local W/O Delay) gives the runtime obtained when these delays were removed. This resulted in a significant reduction from 36 seconds to 22 seconds.

7.3 Summary and Evaluation of Tests

The test results support the following claims:

1. Structure-and-behavior models can be developed for use by a model-based reasoner for electrical power system FDIR;

2. These models are robust enough to accurately simulate the behavior of simple power systems;

3. A model-based diagnoser with the appropriate model, such as the IPC prototype, is capable of correctly monitoring a power system, diagnosing and isolating any electrical faults, and undertake action to cause power flow to be restored to critical, redundantly-wired loads in a short period of time;

4. The IPC prototype can potentially react fast enough to be useful in a real-time application; and

5. The TTA results confirm that the IPC could reside as a ground-based assistant to mission control engineers performing FDIR on spacecraft, but would probably be better suited as an *on-board* assistant to flight personnel.

The time-to-action results are in need of further improvement. Since no special effort was made to optimize the IPC prototype code beyond the initial translation, we are optimistic that the goal of a time-to-action of less than one second can be achieved through further algorithmic and data structure efficiency improvements. From the test results, it is clear that work in this area should focus on the diagnosis and isolation portion of the system, since the times required to perform this function were significantly larger than for the recovery. Lastly, the use of faster platforms, such as a Sparc-10, will most certainly help in this matter.

Overall, the developers consider the data summarized in this section significant. However, the tests also served to shed light on several improvements that need to be made to the prototype in order to commercialize the technology. These are considered in the final section.

8 Summary and Reflections

The success of any project can be measured in terms of both results obtained and knowledge gained. The concrete results obtained, summarized in the previous section, confirmed, in the minds of the developers of the IPC, that a first principles-based approach to autonomous FDIR for spacecraft power systems is feasible. With the right enhancements, a system like the the IPC can be used to effectively maintain and control power distribution for power systems of the size envisioned for future, long mission spacecraft. As the result of this effort, the developers recommend that these enhancements consist of one or more of the following:

1. More effective storage management for the knowledge-base, in order to speed up its access by the reasoner during diagnosis;

2. A more robust knowledge representation, perhaps consisting of one of the following extensions:
    - Developing models of finer granularity, including, for example, a more detailed model of different loads;
    - Incorporation of component fault models;
    - Acquisition of expert knowledge to supplement the first principles-based model, thus resulting in a hybrid system;
    - Multiple models based on either structural or behavioral abstraction;
3. Caching of knowledge obtained during the reasoning process, e.g. through applying techniques like truth maintenance (TMS).

Some of these enhancements are minor; others more substantial. But none imply an abandonment, or even a major change to, the first-principle-based approach. Future extensions to the work of the IPC developers will consist of implementing some of these enhancements.

As noted at the outset, the dual requirements of speed and robustness for the IPC diagnoser was the major challenge to the success of this project, since these requirements tend to be mutually inhibiting. To achieve an adequate degree of success in meeting both requirements, we were led to an approach which relied less on generic knowledge of power system structure and behavior, and more on knowledge about the PMAD itself. On the modeling side, we attained an adequate degree of robustness at the expense of generality: our approach might not easily generalize to power systems which are larger in size and complexity than the PMAD. This concern will be tested by future research, when other power system models will be built and tested on the IPC. On the reasoning side, we were able to attain a greater amount of TTA speed by replacing a general reasoning strategy, constraint suspension, with one that, again, relied more on the properties specific to PMAD.

It seems to us, then, that there are two approaches one can take to developing systems, such as the IPC, which construct and apply models for diagnosis of complex

systems. One approach is to impart to the diagnostic system a more expressive knowledge representation and reasoning device than was originally proposed by researchers in model-based reasoning, who focused their attention on less complex devices. The other approach, which was taken here, is to augment the original structure and behavior representation (as implemented in KATE) by different kinds of knowledge about components. This knowledge may be classified as *heuristic* by some, although we prefer not to use this term, since it is commonly associated with purely experiential knowledge of the expert, whereas the knowledge we incorporated may be a bit more generic. We chose not to incorporate fault models into the system, although, if this second alternative is to be developed further, this might also be considered.

Our current intent is to develop the first alternative further, to expand upon the original model-based knowledge representation for dealing with more complex devices. We currently feel that solving the problem of developing structure and behavior models for complex systems such as power systems will involve applying the idea of *behavioral abstraction.* Recall that the behavioral complexity of a system is measured mathematically in terms of its algebraic transfer function, and physically in terms of the complexity of causal dependency relationships among the components. We hypothesize that this complexity can be managed by abstracting on the basis of different *aspects* of the system. By aspects (of electrical power) we mean things like voltage, current, state (open, closed or tripped), impedance, temperature, etc. They are things that are outputs of or inputs to the device. The circuit equations provide examples of what we term *aspect models.* Each aspect model provides a complete description of one aspect of the system's behavior. The dependency relationships among components within an aspect model are invariably simpler than among components within the complete model (i.e., the model within which all the aspects are represented). The complete model can be recovered by running each model in parallel, noting that the aspect models mutually constrain one another behaviorally. Thus, aspect models provide a modular representation of behavioral complexity. Developing this extension to the first-principles representation of knowledge is a current focus of our research.

Figure 15: Main Menu Bar

9 Appendix: Using the IPC

This appendix provides a comprehensive guide to all functions available to the user, as well as a brief guide for future IPC programmers.

9.1 Starting the IPC and Using the Menus

To start the IPC, just type the name of the executable file, viz. **ipcrt**.

The IPC has several menu options that provide easy control of the runtime environment, and for model modifications. The following disucssion summarizes the functions of each menu option.

The **main menu**, as shown in Figure 15, consists of the following options:

- The **File** button, located on the left-most upper corner of the window provides the following selections:
  - **Exit** - if a runtime session is active, it will terminate the session,
  - **Save Model** - a copy of the original frame files being used will be saved, and the current model will be saved into the same files from which they were loaded.
  - **Load Model** - this selection performs the following:
    1. if changes to the current model have been made but have not been saved, the user will be prompted to optionally save the current model, and
    2. the user will be prompted for the file(s) that are to be loaded. Normal **Unix** pattern matching is accepted.
- The **mode** button provides two options that changes the operation of the IPC. By default, the system is in **Edit** mode. When a **runtime** session is started, it

automatically switches to runtime mode, thus locking out all editing functions. The user has the option of switching between modes during a runtime session.

  - **Edit Model** - the default mode that provides CAD-like operations.
  - **Runtime Mode** - returns to the runtime mode of operation.

- There are several **runtime control** functions available under this menu:

  - **Start IPC** - this selection spawns the IPC process, and switches from the **Edit** mode to the **Runtime** mode.
  - **Restart IPC** - this selection will terminate the current IPC process, and respawn it.
  - **Terminate IPC** - this selection terminates the runtime session.
  - **Show Command Window**
  - **Show Main Window**
  - **Show Recovery Window**
  - **Show Warning Window**

    These selections bring their respective windows to the foreground. Often these windows can be obscured by other windows that are active, and this provides an easy way of accessing them.

- **Update Sensor Values** is a runtime function that sends the *display measurements* command to the IPC. This provides an easy way of updating the screen without direct user interaction with the IPC through the **Command Window.**

Each object on the display has two menus associated with it, depending on the mode of the system. Figure 16 shows the menu available while in the *Edit Mode.*

- The **Edit Icon(s)** selection will display a dialog box which provides the user the ability to modify the icons and their attributes.

- The **Edit Connection(s)** selection will display a dialog box which provides the user the ability to modify the connections and attributes of the object. See Section *Edit Connections Dialog Box* for details.

- **Hide Object and Connection(s)** This selection will cause the object to not be displayed, as well as any connections associated with it.

When the user is in the *Runtime Mode*, there is a different menu available, as shown below in Figure 17.

- The **Maintain** selection will issue a command to the IPC to maintain this object. The user will also be prompted for a value to maintain the object at. An object that is to be maintained will have a green border around it.

- The **Unmaintain** selection will issue a command to the IPC to remove the object from its maintain list. The border of the object will also return to the color of the background.

- The **Unfail** selection will issue a command to the IPC to remove the object from its failed list. The border of the object will also return to the color of the background.

The Edit Icons Dialog Box (Figure 18) depicts the dialog box the user is given. This provides an interactive method way of modifying the icons and their attributes. With this dialog box, one is able to add, delete, and modify icons and their attributes. The list box at the top of the figure shows the currently defined icons for this object. They are labelled **Closed Switch** and **Open Switch**. If a line is selected within this list, the values in the attribute fields will change to correspond to each icon. The currently selected icon definition is the **Closed Switch**. Beneath this list box is a pulldown list box called **Available Icons** which provides a list of icons as defined in the database file. There is also a button labelled **ADD TO LIST** which will put the currently selected icon in the **Available Icons** list into the upper definitions box. Some default values are supplied for a new entry. Beneath the list box are other fields that are related to this object definition:

- **Default Icon**. If checked, this button indicates that this icon is the default icon displayed when the model is first loaded. If there is no default defined, the **Unknown** icon is used.

- **Upper Bound,Lower Bound.** These two fields define the upper and lower bounds for a **Value Driven** icon. **Value Driven** icons allow the user to define multiple icons, which will change during a runtime session. This can simulate a "**tank**" as well, if enough entries are defined.

- The field **Control Function Value** is for state driven objects, and contains the value to be sent to the IPC process, in order to proceed to the next state.

- **Caption.** This field contains the text that is to be displayed near the icon, according to the orientation specified.

- **Foreground Color,Background Color.** These two menus provide the user with colors that can be used in displaying the caption of the selected icon.

- **Label Orientation.** The four selections available are left, right, above, and below, which indicate where the caption should be placed relative to the icon itself. The default placement is below the icon.

- **Preview Icon Layout.** This button will allow the user to see what their definition will look like, in a separate window, without making the changes permanent.

Also, at the bottom of the list box are two other buttons:

- **Accept Changes.** Selecting this button will save the changes made, and will close the dialog box.

- **Cancel** Selecting this button will discard any changes made, and will close the dialog box.

Figure 19 depicts the **Edit Connections** dialog box. This menu provides an interactive method of modifying the objects' connections and their attributes. The topmost list box contains the connections define for the currently selected object. Each connection has the following attributes:

- **Connection Visible.** This boolean field indicates if the user wants the selected connection to be seen on the display.

- **Connection Width.** The user may change the width of the drawn line of a connection with this field. A slide bar is also provided for input. A connection can be 1 to 10 pixels in width.

- **Color of Line.** The user may change the color of the drawn line of a connection with this list box. The current colormap is read into this list, thus some entries have numerical entries instead of text.

Also, at the bottom of the list box are two other buttons:

- **Accept Changes.** Selecting this button will save the changes made, and will close the dialog box.

- **Cancel.** Selecting this button will discard any changes made, and will close the dialog box.

9.2 Building IPC/RT

This section provides future IPC/RT programmers with information about building the IPC/RT, and possible problem areas discovered.

In order to build this system, you need access to the **SpiderWeb** utilities, and have the public domain **XPM** libraries available. You also need a C++ compiler, preferably **GNU**, but others may work as well. Currently the IPC/RT is only guaranteed to work with the GNU C++ compiler version 2.4.5.

There is a make file provided with the source code. Once located inside the source code directory, type: **make**. The system compiles cleanly on a Sparc 2, and a Sparc 10 with g++.

It should be noted that although it is possible to automatically send the **display measurements** command to the **IPC**, it is presently not advisable. The IPC reads the current measurements from the hardware whenever it receives the **display measurements** command. The problem lies in the amount of time it take for the **IPC** to retrieve the measurements from the hardware, and the time it takes to send this information to the **IPCRT**. During this time interval, the IPC is not able to monitor the system.

One suggestion has been made to not ask for the current measurements of the testbed, but to actually send only the last poll taken by the **IPC**. This would solve this problem, but communication delays are still introduced by the request.

Optimally, the **IPC** should automate this by sending only the changes that have occurred since the last request, instead of the whole system. Once this is done, the **RUNTIME:Update Sensor Values** button can be removed from the menu bar.

References

[1] Anderson, P. "Space Station Common Module Network Topology and Hardware Environment", Martin Marietta Astronomics Group, Final Report, 1990.

[2] Chu, B. "Representing binary relations at multiple levels of abstraction" in *Methodologies for Intelligent Systems, 4.* Z. Ras (ed), North-Holland, 1989.

[3] Dague, P., Deves, P., Luciani, P., and Tallibert, P., "Analog System Diagnosis", reprinted in (Hamscher, 1991).

[4] Davis, R. "Diagnostic Reasoning Based on Structure and Behavior" *Artificial Intelligence*, Vol. 24, No.3, 1984, pp. 347-410.

[5] Davis, R., and Hamscher, W., "Model-based Reasoning: Troubleshooting", reprinted in [17], pp. 3-24.

[6] deKleer, J. and Williams, B.C., "Diagnosing Multiple Faults" *Artificial Intelligence*, Vol. 32, No.1, 1987, pp. 97-130.

[7] deKleer, J., "Focusing on Probable Diagnosis", in Hamscher, W., Console, L. and deKleer, J., *Readings in Model-based Diagnosis*, Morgan Kaufmann Publishers, 1992, pp. 131-137.

[8] de Kleer, J,, Mackworth, A. K., Reiter, R., "Characterizing Diagnoses and Faults", *Artificial Intelligence*, 56, 1992, pp. 197-221.

[9] Director, S.W., *Circuit Theory: A Computational Approach.* John Wiley and Sons, 1975.

[10] Dugal-Whitehead, N. R., "The Continuing Development of Poweer System Automation Knowledge", *Proceedings of the Intersociety Energy Conversion Engineering Conference*, 1993.

[11] Dugal-Whitehead, N. R., "Results of an Electical Power System Fault Study", CDDF Final Report No. N06, NASA Tech Paper 3413.

[12] Fesq, L., Stephan, A., and McNamee, L., "Modeling Power Systems for Diagnosis: How Good is Good Enough?", *Proceedings of the 27th Intersociety Energy Conversion Engineering Conference*, San Diego, CA, 1992, Vol. 1, pp. 203-208.

[13] Genesereth, M., "The Use of Design Descriptions in Automated Diagnosis", *Artificial Intellignce*, 24, 1, 1984.

[14] Gholdston, E. W., Janik, D. F., and Lane, G., "A Diagnostic Expert System for Space-based Electrical Power Networks", *Proceedings of the Intersociety Energy Conversion Engineering Conference*, 1988.

[15] Giunchiglia, F. and Walsh, T., "A Theory of Abstraction", *Artificial Intelligence*, 57,2-3, 1992, pp. 323-390.

[16] Gonzalez, A. J., Osborne, R. L., Kemper, C., and Lowenfeld, S., "On-line diagnosis of turbine-generators using artificial intelligence", *IEEE Transactions on Energy Conversion*, Volume EC-1, Number 2, June 1986, pp. 68-74.

[17] Hamscher, W., Console, L., de Kleer, J., eds. *Readings in Model-Based Diagnosis.* Morgan Kaufmann, 1991.

[18] Hester, T., "FIES-II: A Real Time Fault Isolation Expert System", *Proceedings of the Intersociety Energy Conversion Engineering Conference*, 1986.

[19] *Knowledge-based Autonomous Test Engineer: Software Description and Project Overview*, NASA Document, 1991.

[20] Kelly, J., "Diagnosis by constraint propagation in math models" in *Workshop Notes from AAAI-90 on Constraint-directed Reasoning*, 1990.

[21] Konolige, K., "Abduction vs. Closure in Causal Theories", *Artificial Intelligence*, 1992, Vol. 53, No. 3.

[22] Kuipers, B. "Qualitative simulation" *Artificial Intelligence*, Vol. 29, No.3, Sept. 1986.

[23] Lackinger, F., aand Nejdl, W., "Diamon: A Model-based Troubleshooter based on Qualitative Reasoning", *IEEE Expert*, February 1993.

[24] Leitch, R. R., Chantler, M. J., Shen, Q., and Coghill, G. M., "A Preliminary Specification Methodology for Model-based Diagnosis", in Working Notes of DX-93, *The Fourth International Workshop on Principles of Diagnosis*, Aberystwyth, Wales, 1993, pp. 11-31.

[25] Leitch, R., Shen, Q., "Finding Faults With Model Based Diagnosis", *Proceedings of the Second International Workshop on the Principles of Diagnosis*, 1991.

[26] Lollar, L., Weeks, D.J., "The autonomously managed power systems laboratory", *Proceedings of the 23th IECEC*, Denver, Colorado, Vol.3, 1988.

[27] Lloyd, B., Park, W., White, J., and Divakaruni, M., "A Generator Expert Monitoring System", *Proceedings of the Conference on Expert System Applications for the Electric Power Industry*, Orlando, FL, June 1989.

[28] Morris, R., Gonzalez, A., et. al., "A model-based fault diagnostic and control system for spacecraft power" in *Proceedings of the 27th IECEC*, San Diego, CA, 1992, VOl.1, pp. 165-170.

[29] Mozetic, I., "Hierarchical Model-based Diagnosis". Reprinted in (Hamscher, 1991).

[30] Ng, H.T."Model-based, Multiple-fault Diagnosis of Dynamic, Continuous Physical Devices". *IEEE Expert*, December 1991, pp. 38-43.

[31] Priest, C. and Wellman, B. "Modeling Bridge Faults for Diagnosis in Electronic Circuits", in Hamscher, W., (ed.) *Working Notes of the First International Workshop on the Principles of Diagnosis*, 1990, pp. 69-74.

[32] Reiter, R., "A Theory of Diagnosis From First Principles", *Artificial Intelligence*, Vol. 32, No.1, 1991, pp. 57-96.

[33] Russell, B. D., and Watson, K., "Power Substation Automation Using Knowledge-Based Systems", *IEEE Transactions on Power Delivery*, October 1987, pp. 1090-1098.

[34] Scarl, E. "Multi-Level Diagnosis in Model-based Reasoning", Boeing Computing Services, Technical Report BCS-G2010-119, 1993.

[35] Spier, R. J., and Liffring, M. E., "Real-Time Expert Systems for Advanced Power Control", *Proceedings of the Interscociety Energy Conversion Engineering Conference*, 1988.

[36] Struss, P., and Dressler, O., "Physical Negation: Introducing Fault Models into the General Diagnostic Engine", *Proceedings IJCAI-89*, Detroit, MI 1989, pp. 1318-1323.

[37] Struss, P. "What's in SD? Towards a theory of modeling for diagnosis", in [17], pp. 419-450.

[38] Watson, K., Russell, B. D., and Hackler, I., "Expert System Structures for Fault Detection in Spaceborne Power Systems", *Proceedings of the Intersociety Energy Conversion Engineering Conference*, 1988.

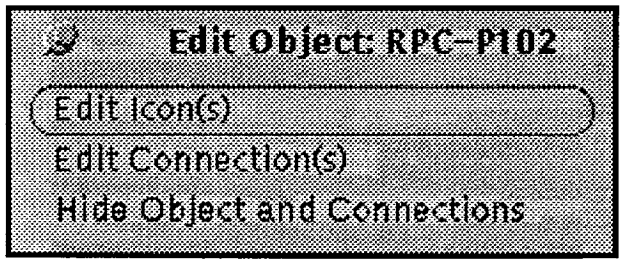

Figure 16: Edit Object Menu

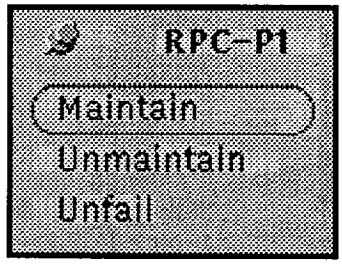

Figure 17: Runtime Object Menu

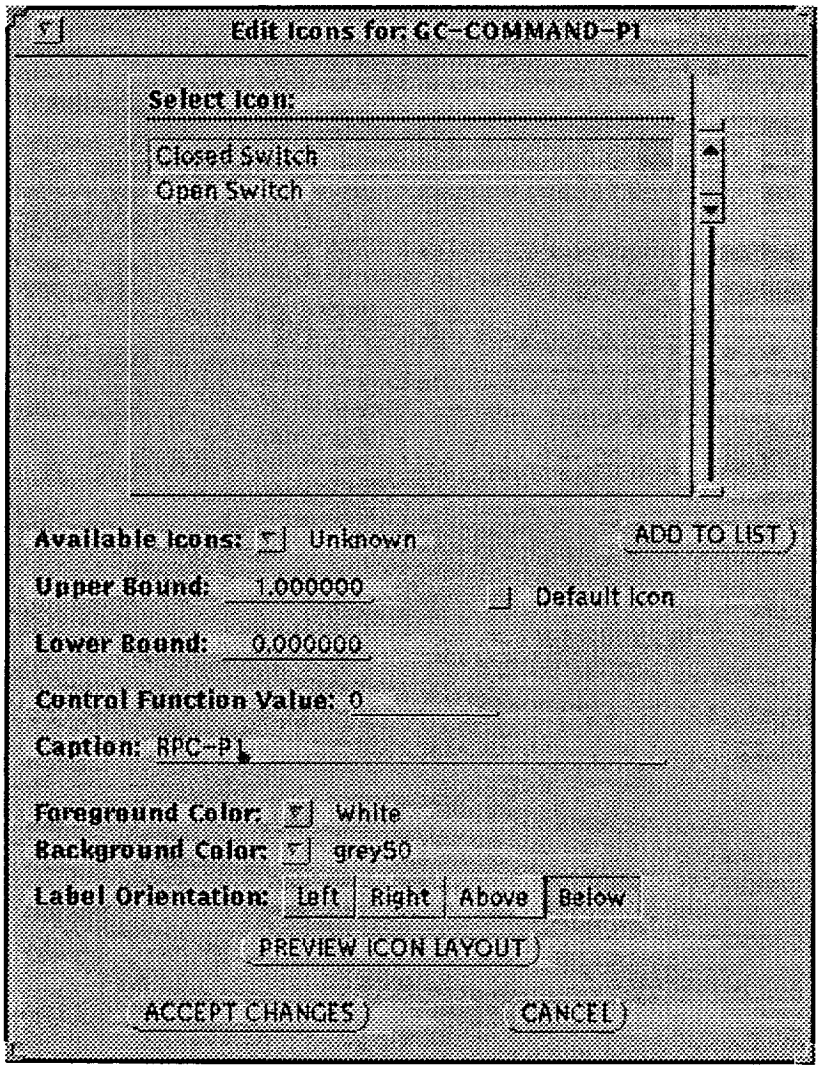

Figure 18: Edit Icons Dialog Box

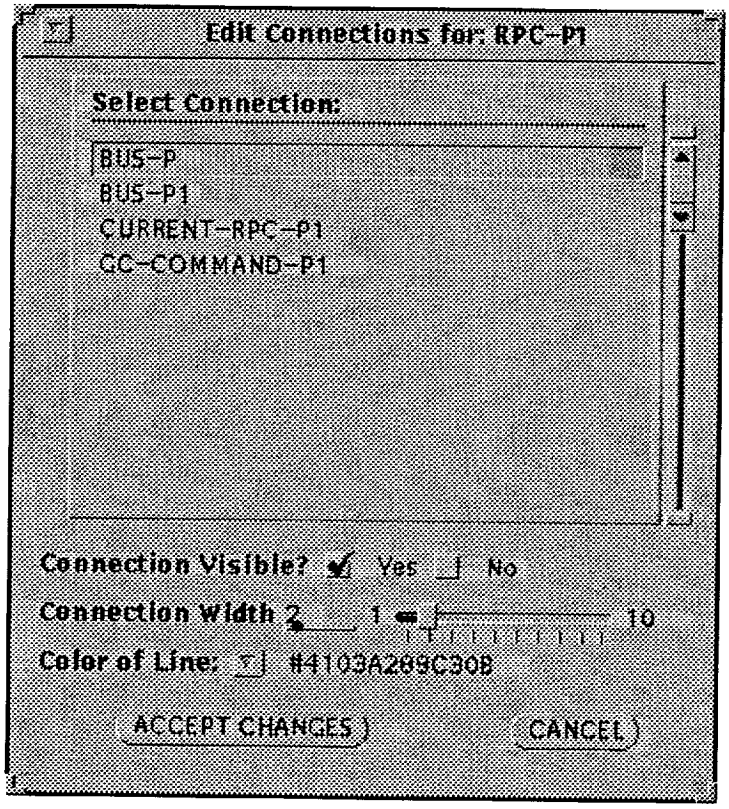

Figure 19: Edit Connections Dialog Box

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November, 1993	Final Report

4. TITLE AND SUBTITLE
MODEL-BASED REASONING FOR POWER SYSTEM MANAGEMENT - USING KATE AND THE SSM-PMAD

5. FUNDING NUMBERS
Contract NAS8-39385

6. AUTHOR(S)
R. Morris, A. Gonzalez, D. Carriera,
F. D. McKenzie, B. Gann

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Florida Institute of Technology
150 W. University Blvd.
Melbourne, FL 32901-6988

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Prepared in cooperation with the University of Central Florida
Orlando, FL 32816

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified
Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
The overall goal of this research effort has been the development of a software system which automates tasks related to monitoring and controlling electrical power distribution in spacecraft electrical power systems. The resulting software system is called the Intelligent Power Controller (IPC). The specific tasks performed by the IPC include continuous monitoring of the flow of power from a source to a set of loads, fast detection of anomalous behavior indicating a fault to one of the components of the distribution systems, generation of diagnosis (explanation) of anomalous behavior, isolation of faulty object from remainder of system and maintenance of flow of power to critical loads and systems (e.g. life-support) despite fault conditions being present (recovery). The IPC system has evolved out of KATE (Knowledge-based Autonomous Test Engineer), developed at NASA-KSC. KATE consists of a set of software tools for developing and applying structure and behavior models to monitoring, diagnostic and control applications.

14. SUBJECT TERMS
electrical power systems, model-based reasoning, fault-detection, isolation, recovery (FDIR)

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	unlimited