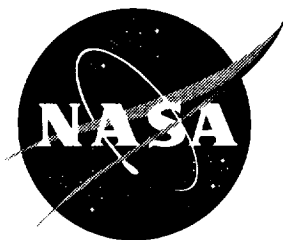


NASA Technical Memorandum 104786

Making Intelligent Systems Team Players: Additional Case Studies

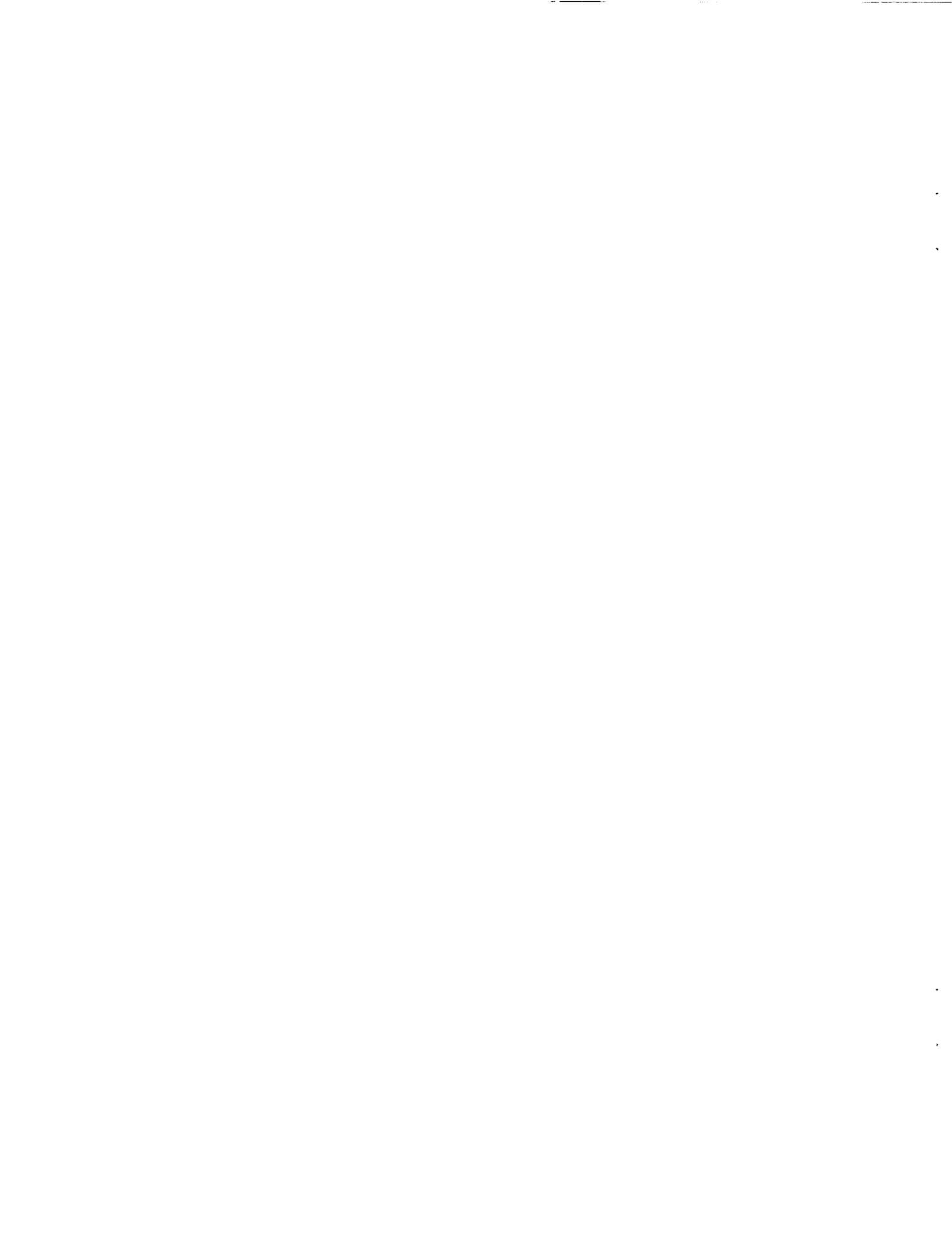
Jane T. Malin
*Lyndon B. Johnson Space Center
Houston, Texas*

Debra L. Schreckenghost
Ron W. Rhoads
*The MITRE Corporation
Houston, Texas*



National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

December 1993



ACKNOWLEDGMENT

We would like to thank all the NASA and contractor personnel who took time to demonstrate and explain their systems to us. Their patience and willingness to share their problems and ideas fostered an ideal environment in which to undertake this work. We also appreciate the time expended by these personnel to review the case study report for accuracy.

CONTENTS

Section		Page
1	INTRODUCTION	1-1
1.1	Purpose and Objectives	1-1
1.2	Approach and Scope	1-1
1.3	Organization of Document	1-2
1.4	References	1-2
2	PROPULSION PROCEDURAL REASONING SYSTEM	2-1
2.1	System Description	2-1
2.2	Intelligent System and Functions	2-2
2.3	Human-Intelligent System Interaction Functions	2-5
2.4	Supporting User Interface Capability	2-7
2.5	Design Process	2-11
2.6	Study Method	2-12
2.7	Case Data Sources	2-12
3	COOPERATING EXPERT SYSTEM	3-1
3.1	System Description	3-1
3.2	Intelligent System and Functions	3-1
3.3	Human-Intelligent System Interaction Functions	3-4
3.4	Supporting User Interface Capabilities	3-5
3.5	Design Process	3-9
3.6	Study Method	3-10
3.7	Case Data Sources	3-10
4	EXTENDED REAL-TIME FEAT	4-1
4.1	System Description	4-1
4.2	Intelligent System and Functions	4-1
4.3	Human-Intelligent System Interaction Functions	4-3
4.4	Supporting User Interface Capabilities	4-5
4.5	Design Process	4-12
4.6	Study Method	4-14
4.7	Case Data Sources	4-14
5	PDRS DECISION SUPPORT SYSTEM	5-1
5.1	System Description	5-1
5.2	Intelligent System and Functions	5-1
5.3	Human-Intelligent System Interaction Functions	5-3
5.4	Supporting User Interface Capability	5-4
5.5	Design Process	5-9
5.6	Study Method	5-9
5.7	Case Data Sources	5-10
6	FUEL CELL MONITORING SYSTEM	6-1
6.1	System Description	6-1
6.2	Intelligent System and Functions	6-1
6.3	Human-Intelligent System Interaction Functions	6-3
6.4	Supporting User Interface Capabilities	6-3
6.5	Design Process	6-7
6.6	Study Method	6-7
6.7	Case Data Sources	6-8

FIGURES

Figure		Page
2-1	Example of a Knowledge Area	2-3
2-2	Example of an Intention Graph	2-4
2-3	Typical Architecture of PRS Instance	2-4
2-4	Workspace of the Procedural Reasoning System	2-8
2-5	Menu Hierarchy of the PRS Control Window	2-9
2-6	Examples of the IO Window, Text Trace Window, and Command Window ..	2-10
3-1	Typical CoopES Configuration	3-3
3-2	The Workspace of CoopES	3-6
3-3	Example of the SM Interface Display	3-7
3-4	Example of the EPSOI Display	3-8
3-5	Example of the RCSOI Display	3-9
3-6	Example of the Configuration Manager Display	3-10
4-1	A System Schematic and its System Digraph Model	4-2
4-2	The Workspace of FEAT	4-5
4-3	Example of the FEAT Display	4-7
4-4	Description of Analysis Capabilities Provided by Action Bar	4-9
4-5	Description of Analysis Capabilities Provided by Menus	4-9
4-6	Icons and Color Coding used in the Overview Window	4-10
4-7	An Example of the Overview Window	4-11
4-8	Data Flow Diagram of the Fault Detection Process	4-13
4-9	Information Exchanged between Flight Controller and ERF	4-14
5-1	Example of a Self-correcting Rule in DESSY	5-3
5-2	Original HCI Design Concepts for the MPM/MRL Subsystem Status Display	5-5
5-3	Information from Integrated Status Display Combined with MPM/MRL Subsystem Status Display	5-6
5-4	Illustration of a Portion of DESSY User Interface after Manual Take Over	5-7
5-5	Example of the DESSY Timer Pop-Up Window	5-8
6-1	FCMS Workspace Layout	6-4
6-2	The Algorithm Control Panel	6-5

TABLES

Table		Page
4-1	Graphic Forms in a Digraph	4-6
4-2	Color Coding Used on the Failure Digraph	4-10
5-1	Color Coding in the PDRS DESSY Displays	5-8

ACRONYMS AND ABBREVIATIONS

AOS	acquisition of signal
AI	artificial intelligence
C&W	caution and warning
CCC	Combined Control Center
CM	Configuration Manager
CoopES	Cooperating Expert System
DESSY	Decision Support System
DFD	data flow diagrams
EPS	Electrical Power System
EECOM	Electrical, Environmental, Consumables, and Mechanical
EpSys	EPS Expert System
ERF	Extended Real-Time FEAT
FEAT	Failure Environment Analysis Tool
FDM	Fault Detection and Management
FCMS	Fuel Cell Monitoring System
FMEA	Failure Modes and Effects Analysis
HCI	human-computer interaction
JSC	Johnson Space Center
KA	knowledge area
LOS	loss of signal
MPM	Manipulator Position Mechanism
MPSR	Multi-Purpose Support Room
MRL	Manipulator Retention Latch
MCC	Mission Control Center
OAST	Office of Aeronautics and Space Technology
OACT	Office of Aeronautics, Commercialization, and Technology
OMS	Orbital Maneuvering System
PDRS	Payload Deployment and Retrieval System
PRS	Procedural Reasoning System
PROP	Propulsion System Engineer
PropSys	RCS expert system
RCS	Reaction Control System
RTDS	Real-Time Data System
RMS	Remote Manipulator System
RTOP	Research and Technology Operating Plan
SM	System Monitor
TOL	telemetry object list

SECTION 1

INTRODUCTION

The use of advanced automation technology in space operations is complicated by the demanding conditions of that environment. These conditions include high risk actions, noisy data, dynamic complex systems, and large quantities of information that must be processed in a timely manner. Designing intelligent systems¹ for effective human-computer interaction (HCI) in such an environment requires defining the information that must be exchanged between the human and the intelligent system to perform tasks (the *information requirements*). These information requirements include both the information supporting the domain tasks (monitoring and controlling a space system) and the information supporting the human in the new task of supervising the intelligent system. Failure to provide either type of information can result in systems that are not robust and are difficult to use.

A series of studies have been conducted to investigate the design of robust, usable intelligent systems for space operations (what we have termed "designing intelligent systems to be team players"). The first study was conducted in 1991. The purpose of this study was to provide guidance in designing intelligent fault management systems for improved human-computer interaction. The results of this study are documented in two NASA technical memoranda (Malin, et al, 1991; Malin and Schreckenghost, 1992). In 1992, additional cases were studied to investigate issues identified in the initial study. This report documents the results of this second case study. Both of these studies were conducted as part of a Research and Technology Operating Plan (RTOP) for the Office of Aeronautics, Commercialization, and Technology (OACT)².

1.1 Purpose and Objectives

The results of this study are intended as a supplement to the original design guidance documents (Malin, et al, 1991; Malin and Schreckenghost, 1992). These results should be of interest to designers of intelligent systems for use in real-time operations, and to researchers in the areas of human-computer interaction and artificial intelligence.

The purpose of studying additional cases was to broaden the investigation of human-computer interaction design issues beyond the focus on monitoring and fault detection in the first case study (Malin and Schreckenghost, 1992). New issues include (1) supporting the supervision of intelligent systems, (2) using shared representation as an alternative to explanation, and (3) integrating and coordinating multiple intelligent systems. The additional cases include intelligent software for failure impact assessment, procedure analysis, and replanning in real-time.

1.2 Approach and Scope

Five systems were selected for study. These systems are summarized below:

- **Propulsion Procedural Reasoning System (PRS):** a fault management system for use by the Space Shuttle propulsion flight controllers in identifying and recovering from faults in the Reaction Control Systems (RCS) and the Orbital Maneuvering Systems (OMS). PRS

¹ An *intelligent system* is a computer system that uses information intelligently to achieve goals. Commercial development environments for intelligent systems often include advanced graphical user interface capabilities.

² Formerly the Office of Aeronautics, Exploration, and Technology (OAET).

dynamically constructs fault diagnosis and correction plans (based on pre-existing malfunction procedures) and monitors the execution of those plans using telemetry data.

- **Cooperating Expert System (CoopES):** a distributed system that demonstrates cooperation among multiple expert systems performing real-time fault management. CoopES consists of two fault management expert systems, and the software necessary for these systems to communicate and coordinate their activities. Both expert systems were developed previously as stand-alone systems for managing failures in the Space Shuttle Electrical Power System (EPS) and the RCS, respectively.
- **Extended Real-Time Failure Environment Analysis Tool (FEAT) (ERF):** a fault management tool developed for both Space Shuttle and Space Station ground flight controllers. ERF assists the flight controller in determining failure causes (fault isolation), failure propagation paths, and failure effects (or impact assessments). ERF extends the capabilities of FEAT by providing automated real-time analysis of failure models using telemetry data.
- **Payload Deployment and Retrieval System (PDRS) Decision Support System (DESSY):** a fault management system for the Space Shuttle Remote Manipulator System (RMS). DESSY assists the PDRS flight controllers in monitoring status and detecting faults for two RMS subsystems, the Manipulator Position Mechanisms (MPMs) and the Manipulator Retention Latches (MRLs). Because DESSY was also part of the first study, a description is included of some problems with the original design and how these problems have been resolved.
- **Fuel Cell Monitoring System (FCMS):** a fault management system for the Space Shuttle fuel cells and associated power busses. The FCMS is used by the Electrical, Environmental, Consumables, and Mechanical Systems (EECOM) flight controllers to assess the status of the fuel cells and power busses, and to recommend crew and flight controller actions based on this assessment.

Information about each system was gathered by interviewing system developers and users, and by reading related papers and documents. Using this information, a case report has been written for each intelligent system in the study. Each case report includes a description of the intelligent system functions and the associated space domain system, an analysis of how the human interacts with the intelligent system during operations, an evaluation of the user interface capabilities provided to support such interaction, and a description of the methodology used to design and build the intelligent system.

1.3 Organization of Document

Section 1 of this report describes the objectives of the second case study and introduces the cases selected for that study. Sections 2 through 6 contain the case reports for each of the five systems studied. A reference list summarizing the documents used in this study is also attached.

1.4 References

Malin, J., D. Schreckenghost, D. Woods, S. Potter, L. Johannesen, M. Holloway, and K. Forbus. *Making Intelligent Systems Team Players: Case Studies and Design Issues. Volume 1. Human-Computer Interaction Design, Volume 2 Fault Management System Cases.* NASA Technical Memorandum 104738. Houston, TX: NASA/Johnson Space Center. October 1991.

Malin, J., and D. Schreckenghost. *Making Intelligent Systems Team Players: Overview for Designers*. NASA Technical Memorandum 104751. Houston, TX: NASA/Johnson Space Center. June 1992.

SECTION 2

PROPULSION PROCEDURAL REASONING SYSTEM

2.1 System Description

The PRS is a tool developed for creating plans to achieve specific goals and monitoring plan execution. The PRS tool has been used to develop a fault management system for use by the Space Shuttle propulsion flight controllers³. The purpose of this system is to assist in identifying and recovering from faults by constructing plans (based on malfunction procedures⁴) and monitoring the execution of those plans. Hardware states and status provided by fault detection software are used to select "pieces" of pre-defined procedures. These pieces of procedures are used to build a plan for isolating and recovering from a specific fault. The plans that are built are communicated by the flight controller to the crew, via the voice loop. The system then monitors the telemetry data for indications that the steps of the plan have been executed. Although a separate fault detection system is planned, the current system includes embedded fault detection.

The PRS tool permits several instances of the basic system to run in parallel. The design for the propulsion system includes six instances of the PRS software running in parallel, corresponding to five instances for managing faults in the propulsion subsystems and one instance for managing the user and data interface for all other instances. The specific propulsion subsystems being managed include (1) the right aft, left aft, and forward RCS, (2) the right and left OMSs⁵. To date, this application deals only with the RCS system and addresses 17 known RCS single failure conditions (corresponding to approximately 10 single malfunction procedures).

The ability of the propulsion application of PRS to generate plans and recommend procedures is unique among the systems in case study. In other cases (such as CoopES, section 3), the only real-time planning activity is selecting among pre-defined plans. PRS supports dynamic replanning by creating new plans on-the-fly from existing "pieces" of procedures. PRS also supports creating new "pieces" of procedures, resulting in completely new procedures. This capability helps flight controllers construct contingency workarounds in real time.

Another unique aspect of PRS is the ability to respond to situations in which multiple faults occur. The ability to manage multiple faults is not typical in diagnostic systems, and is not included in Space Shuttle malfunction procedures. This application of PRS can respond to multiple faults by (1) having multiple fault management tasks in progress (although executed sequentially) for a single instance of PRS, and (2) running multiple instances of PRS in parallel. The system can also assist the flight controller in creating new procedures for multiple faults (e.g., altering procedures to accommodate fault interaction).

The propulsion application is built using a version of PRS written in Common LISP on a Sun workstation. This version of PRS communicates using TCP/IP and has an XWindows interface. A port to the newest version of PRS, written in C with a Motif style interface, is planned in the near future. This port is expected to solve some of the real-time performance problems seen with

³ The propulsion application of PRS is planned for use by both the Propulsion System Engineer (PROP) the Flight Control Room and the PROP controllers in the Multi-Purpose Support Room (MPSR).

⁴ Throughout this report, a *procedure* refers to a plan template that includes possible courses of action in operational situations (e.g., a malfunction procedure addresses fault situations). A *plan* refers to a single course of action, based on relevant procedures and selected to address a specific set of conditions.

⁵ The RCS modules provide low levels of thrust for controlling the attitude of the Space Shuttle. The OMS provide much higher levels of thrust for maneuvering the vehicle.

the LISP version. Moreover, LISP is not approved for the Mission Control Center (MCC) workstations.

2.2 Intelligent System and Functions

The propulsion application has a multi-agent architecture. It consists of six instances of PRS (or agents) that run in parallel. Five of these instances manage faults in the propulsion subsystems (RCS, OMS). One instance manages the interface to the data and the user, including (1) handling sensor data, (2) recommending control actions using effectors, and (3) checking for failed sensors and effectors. Although each instance has a separate meta-level reasoner that coordinates its activities, only the human supervises the interaction of instances. Instances can influence each other by changing facts in the shared database, or by directly communicating. Instances operate asynchronously and communicate by message passing (e.g., to request activation of a knowledge area [KA] or goal). Response to a request by an instance depends on the reliability of the requester, the type of message, and the state of the receiving instance (its beliefs, goals, and intentions). Most of the interaction among these instances occurs with the interface instance. The subsystem instances operate in relative isolation (each has its own goals, KAs, intention graph, and meta-level reasoner).

An instance of PRS consists of a database of facts and beliefs about the monitored process, a set of goals to be achieved, a set of plans used to achieve these goals, an intention structure containing all plans in progress, and an interpreter to construct the intention structure and to handle prioritization of plans and changes in priority. A PRS instance operating in real time includes the following:

- **Database**
The database is the shared memory for all PRS instances. It contains known facts based on telemetry or user input, and the current beliefs of the system. The database includes state and status descriptions as well as structural information. Information in the database is provided by the data server (transforms numeric telemetry data to symbolic values) or the user, or is derived by an instance. Facts and beliefs are updated by the system as available. Values in this database trigger the selection of plans.
- **Goals**
Goals are descriptions of desired system behaviors or crew tasks. Every goal has an associated set of actions derived from a piece of a malfunction procedure (called a knowledge area) that is explicitly declared to be the achiever of that goal. A goal is successfully achieved when the set of actions is successfully completed. As goals are selected, they may also be tested (i.e., determine if goal has already been met), maintained (i.e., held constant while other tasks are executed), or placed in a waiting state (i.e., delayed until specific conditions become true).
- **Knowledge Area**
A KA in PRS is a set of actions for achieving a specific goal. For this application, KAs represent the pieces of malfunction procedures used to construct plans. Each KA is represented as a frame consisting of a body, an invocation, a context, and optional slots. The body specifies procedure steps (an example of a KA is shown in figure 2-1; procedure steps are on the arcs of the graphic). A KA associated with an active goal is executed when both the events in the invocation slot occur and the conditions in the context slot are true of the current state. Use of the optional slots (effects, goal achiever, properties) vary depending on what items are needed for execution.

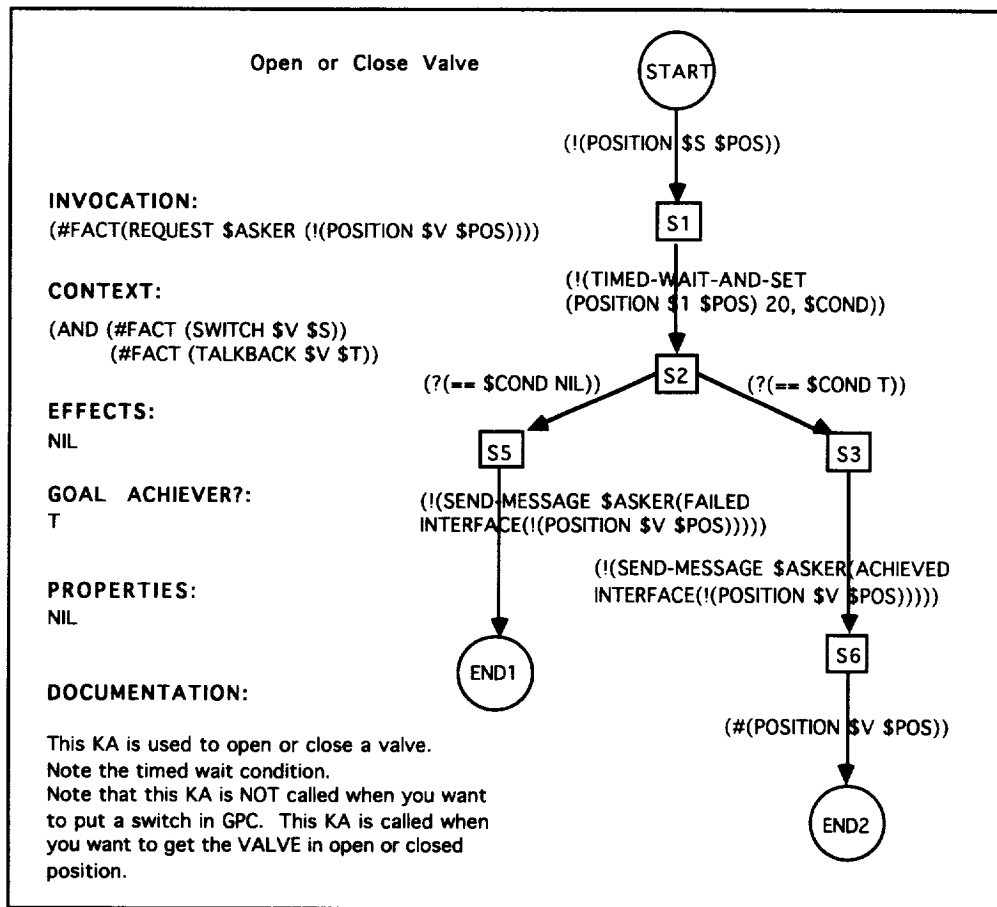


Figure 2-1. Example of a Knowledge Area.

- **Intention Graph**

The intention graph represents the tasks that the system expects to be executed (either immediately or later). The term *task* refers to the plans (KAs) selected by the System Interpreter to achieve an active goal. Successful completion of a task results in achievement of the goal. At any given time, an intended task may be executing, waiting for activation conditions, or delayed (suspended or deferred). If a more urgent goal arises, the ongoing task can be suspended and a task activated to achieve the new goal. Figure 2-2 illustrates an intention graph.

- **Interpreter**

The Interpreter is a meta-level reasoner that uses system beliefs and goals to select plans (KAs) and to initiate plan monitoring for the tasks specified in the intention structure. As plans are executed, and mission operations continue, state changes result in new goals being set and new beliefs being derived. The Interpreter manages changes to the intention structure resulting from these changes in beliefs and goals. It also determines the priority of pending tasks (which task to work on next) based on these changes.

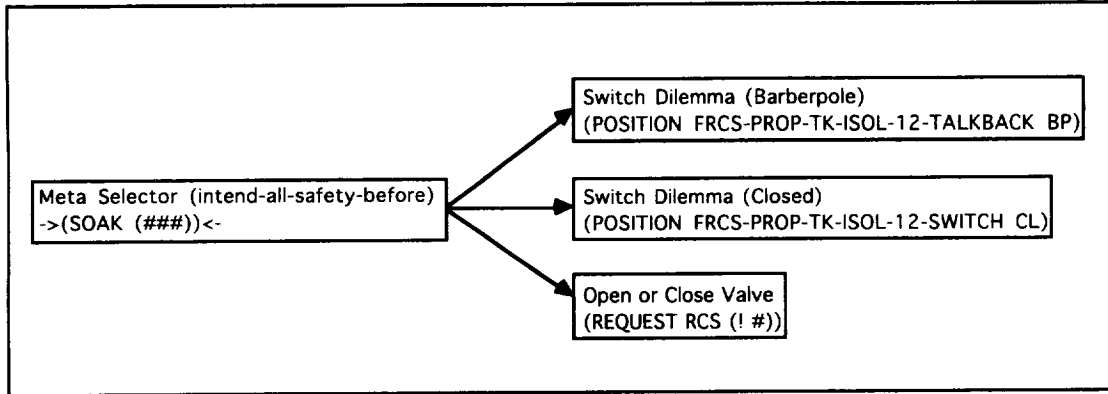


Figure 2-2. Example of an Intention Graph.

Figure 2-3 illustrates the typical architecture for a single PRS instance. The parts of the system in the shaded boxes are active during operational use of the system. The non-shaded portions of this figure illustrate how the PRS is configured for software development and testing.

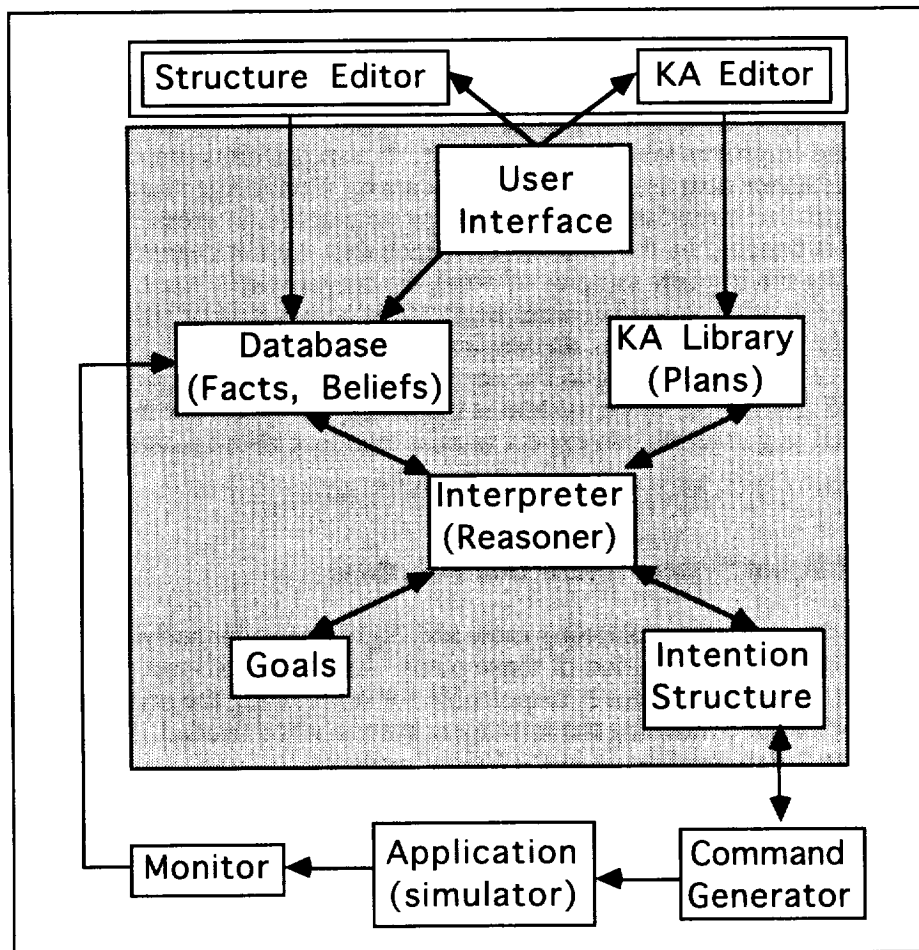


Figure 2-3. Typical Architecture of PRS Instance.

The intention graph represents the composite set of plans built to achieve all active goals. This graph is made up of multiple planned tasks, each associated with a goal. Although multiple tasks can be pending, only one task (and associated KA) is executed at a time for an instance of PRS. Executing a KA is equivalent to monitoring for evidence in data that the steps of the recommended plan are being executed. A KA is executed by evaluating the logic expression associated with each step of the plan. If the logic expression evaluates true, the plan step has been successfully completed and the system begins monitoring the next step (moves down the KA graph in figure 2-1). If the logic expression evaluates false, the plan step has failed, and an alternative path is tried (move up the KA graph in figure 2-1). To accommodate for steps executed out of order, PRS checks whether goals have been satisfied before moving to the next plan step. To keep the user informed of the situation (success or failure), user status messages are included in each KA.

PRS is a reactive system that can respond to situations that develop in unexpected ways (where expectations are based on the malfunction procedures). For example, plan steps do not have to be executed in an exact order, as long as the goal is achieved. To accommodate changes in situation (as represented in goals and beliefs), a KA being executed can be (1) interrupted by a new KA corresponding to a higher priority goal⁶, (2) suspended until specific conditions apply, and (3) aborted when necessary. The intention graph is changed if a new task (and goal) takes precedence over an executing task. The original task (i.e., all the KAs required to achieve a goal) will be placed in a wait state until the higher priority task finishes. When the higher priority task is completed, the original task will be re-evaluated before resuming activity, to ensure that it is still valid and necessary. If it is not needed, it is deleted.

Telemetry data are processed prior to use in PRS. A data server converts numeric data to symbolic values. Since only changes are transmitted to PRS, this reduces the amount of information that must be processed and improves system performance. It also permits compensating for noisy data. The Interface instance also assists in handling data by monitoring sensor status and transforming raw data (transducer readings) into needed parameters (e.g., pressure). Loss of data at loss of signal (LOS) is handled by the data server. Since data are not changing during LOS (except for cyclically repeating the last few samples of data), the server is quiescent, and PRS waits to resume processing when information becomes available at acquisition of signal (AOS). If monitoring a plan with timed operations that expire during LOS, however, PRS either aborts the process being timed or issues a warning to the operator about the timer expiring. One of the drawbacks of making LOS transparent to the system is that it may be useful for the system to recognize that LOS has occurred (e.g., system can expect to miss evidence about plan execution during LOS).

2.3 Human-Intelligent System Interaction Functions

PRS performs two basic activities: planning crew and flight controller activities in response to anomalies, and monitoring the execution of those plans. Although PRS can perform these activities autonomously, the human operator is responsible for supervising the accomplishment of these activities, and redirecting or overriding the intelligent system when needed. The operator can intervene in planning and plan monitoring in the following ways:

- Specify the priority of a task in the intention graph (and thus the order in which tasks are executed)
- Select among a set of valid plans (KAs)
- Add new goals
- Alter facts used by PRS
- Delete parts of the plan by removing a KA from intention graph

⁶ Goal priorities are predefined, but can be changed by the flight controller in real time.

- Abort, restart, or suspend⁷ execution of a KA
- Shut down an instance of PRS

To assist the operator in performing these activities, PRS provides information about goals (and their priorities), plans or KAs, tasks in intention graph (including task ordering), messages about status of plan execution, facts, and system beliefs. The use of this information and capability when interacting with the intelligent system is discussed below.

PRS supports flight controllers in replanning during a mission. Although the system automatically replans when goals or beliefs change, the operator can influence this replanning process. The operator can directly change goals used to select plans, or can provide information about the monitored process that changes beliefs and goals. The operator can alter the priority of goals, resulting in changes in task ordering on the intention graph. If PRS generates more than one valid plan, the operator can select among plans. The operator can also override the default plan (a form of manual replanning) by aborting, restarting, suspending, or deleting a KA on the intention graph.

Contingency situations require replanning with special workaround procedures. In a contingency situation, existing malfunction procedures (and the KAs constructed from them) cannot resolve the anomaly in the monitored process. New procedures must be constructed during the mission. PRS supports responding to contingency situations by permitting the operator to generate new KAs (corresponding to new workaround procedures) on-the-fly and use them immediately in planning.

To monitor the execution of the plan it has suggested, PRS looks for evidence in the telemetry data that each step of the plan (KA) has been executed. If the expected evidence is found, the operator is informed about the success of the activity. If data indicates a deviation from the plan, the operator is notified. If there is no evidence in the data that the crew has begun executing the plan, the system reminds the operator that it is "still waiting" (after a reasonable amount of time). The operator can respond to either of these problem situations by notifying the crew of the problem, or by altering or interrupting the plan (abort, restart, suspend, or delete a KA; change a goal). If the operator takes no action, PRS continues to monitor the selected plan until data indicates that the associated goal has been reached, or that changes have occurred requiring a new plan. One of the difficulties of plan monitoring is that plans are not always executed as specified (e.g., reverse the order of steps, if order is not critical). PRS accommodates such situations by detecting and notifying the operator when an active goal is satisfied, and removing the appropriate KA from the intention graph.

Each PRS instance supports the operator in supervising its activities. It provides the operator with visibility into its knowledge base by displaying KAs (presented graphically or in text) and the intention graph. The operator can also search the knowledge base to identify which KAs might respond to a specified goal change or fact change. The results of planning activities are reflected in the intention graph, which can be monitored while the system runs. To keep the operator informed about plan monitoring activities, the intelligent system sends messages describing the plan step it is currently monitoring (i.e., its current hypothesis) and why it is concerned with this plan (i.e., the goal that will be achieved if the plan executes successfully).

To assist in supervising intelligent system activities, a representation of the plans of each instance is shared with the human supervisor. The human monitors this representation as plans are executed. This assists the operator in establishing expectations about what will happen in the monitored process (based on the plan) and in understanding situations as they develop. It also illustrates the ongoing activities of each instance in the context of the monitored process situation. This design has some limitations, however. The data used to draw conclusions are not evident,

⁷ The interval of suspension can be delimited either by time or event.

and there is no integrated representation of plans across instances, which means there is no support for understanding the overall situation and how instances interact in response to that situation.

Each PRS instance also supports the operator in intervening in its activities. The operator can interrupt intelligent system monitoring activities by suspending a KA. The operator can redirect the intelligent system (1) by informing the system of new goals, new facts, or even new KAs that affect its planning activities, or (2) by manually changing the plan to alter its plan monitoring activities. The operator can even selectively override the intelligent system when it is in error. The knowledge base of this application is partitioned into six instances of PRS, one each for the five subsystems of propulsion system and one for handling the data interface. Shutting down an instance of PRS is equivalent to disabling a portion of the knowledge base. This requires that the operator manually take over the planning and plan monitoring activities for that subsystem.

PRS provides limited support to the operator in supervising multiple instances. Managing six instances requires monitoring six sets of information. There is no way of monitoring the activities of all instances at one time, and interaction among instances is not clearly represented. As a result, it is difficult for the operator to gain an understanding of the overall situation and to track parallel activities and changes in those activities. To support human supervision of multiple instances, information from these different instances must be integrated for use by a single operator (e.g., merge plan information into a single intention graph). Important relationships among this information must be identified for the operator (e.g., failures that affect multiple systems, such as power failures) and the overall situation must be represented, including interaction among instances. The operator must be able to effectively monitor and manage multiple lines of reasoning in parallel (e.g., keep track of which instance provides information, or which instance has been informed of a fact; shift focus of attention among instances). These type of issues were being considered by the design team at the time of the interviews.

2.4 Supporting User Interface Capability

User interface design for the propulsion application of the PRS was just beginning when the case interviews were done. At that time, the PRS development user interface was the only user interface design available. Although the final design will change, it is likely that much of the PRS development user interface design will be retained in the final design. Thus, the PRS development user interface is described in this section, particularly how it supports the operator in managing the monitored process and in supervising the intelligent system. This discussion focuses on the operational use of the PRS user interface instead of its use for software development and testing.

The workspace layout for the PRS development tool consists of the following display areas:

- **Control Window:** provides the user with capability to control the system and configure the user interface
- **IO Window:** posts messages from the intelligent system
- **Text Trace Window:** shows messages and graphics about plan execution
- **Command Window:** permits text-based user input to the intelligent system

Each of these areas is discussed below. Figure 2-4 shows the workspace of the PRS development user interface.

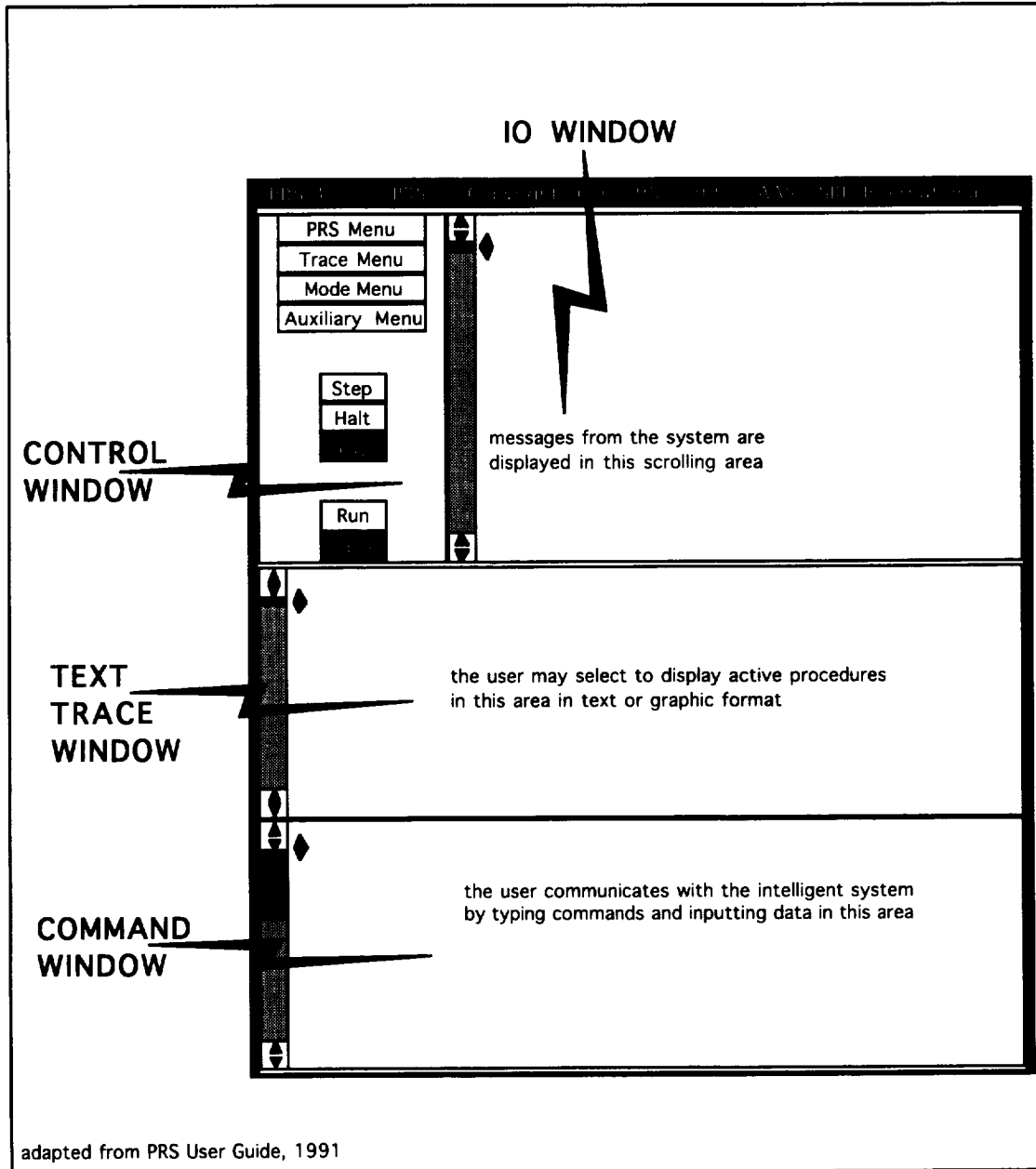


Figure 2-4. Workspace of the Procedural Reasoning System.

The Control Window is located in the upper left of the workspace. This area provides mouse-selectable menus for controlling and configuring PRS. Control capability used during operations includes (1) loading and running the default application, (2) controlling instances of the PRS (start, stop, reset), (3) monitoring the intention graph, KAs, and instances, and (4) adding new goals, facts, and KAs. Configuration capability includes (1) selecting a PRS instance to be associated with the user interface, (2) specifying the display mode for the procedure trace, and (3) selecting the input mode for the Command Window. Capability provided by the menu items is shown in figure 2-5.

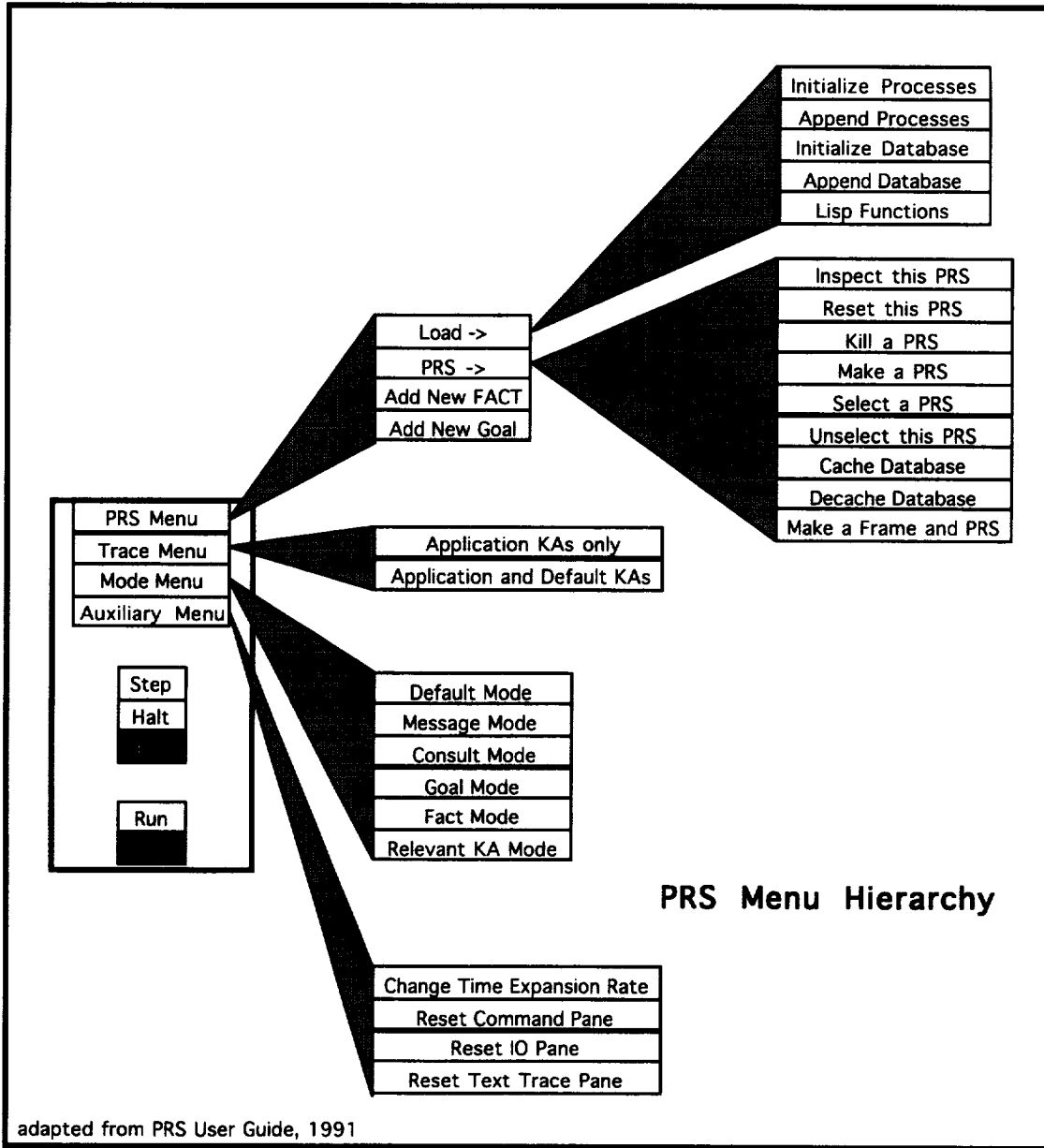


Figure 2-5. Menu Hierarchy of the PRS Control Window.

The IO Window is located in the upper right of the workspace. Input/output messages from the intelligent system are posted in this scrolling window. These messages include requests for information from the operator, suggestions for procedures to be performed, and warning messages about violations of procedure steps. See figure 2-6 for an example of the IO Window.

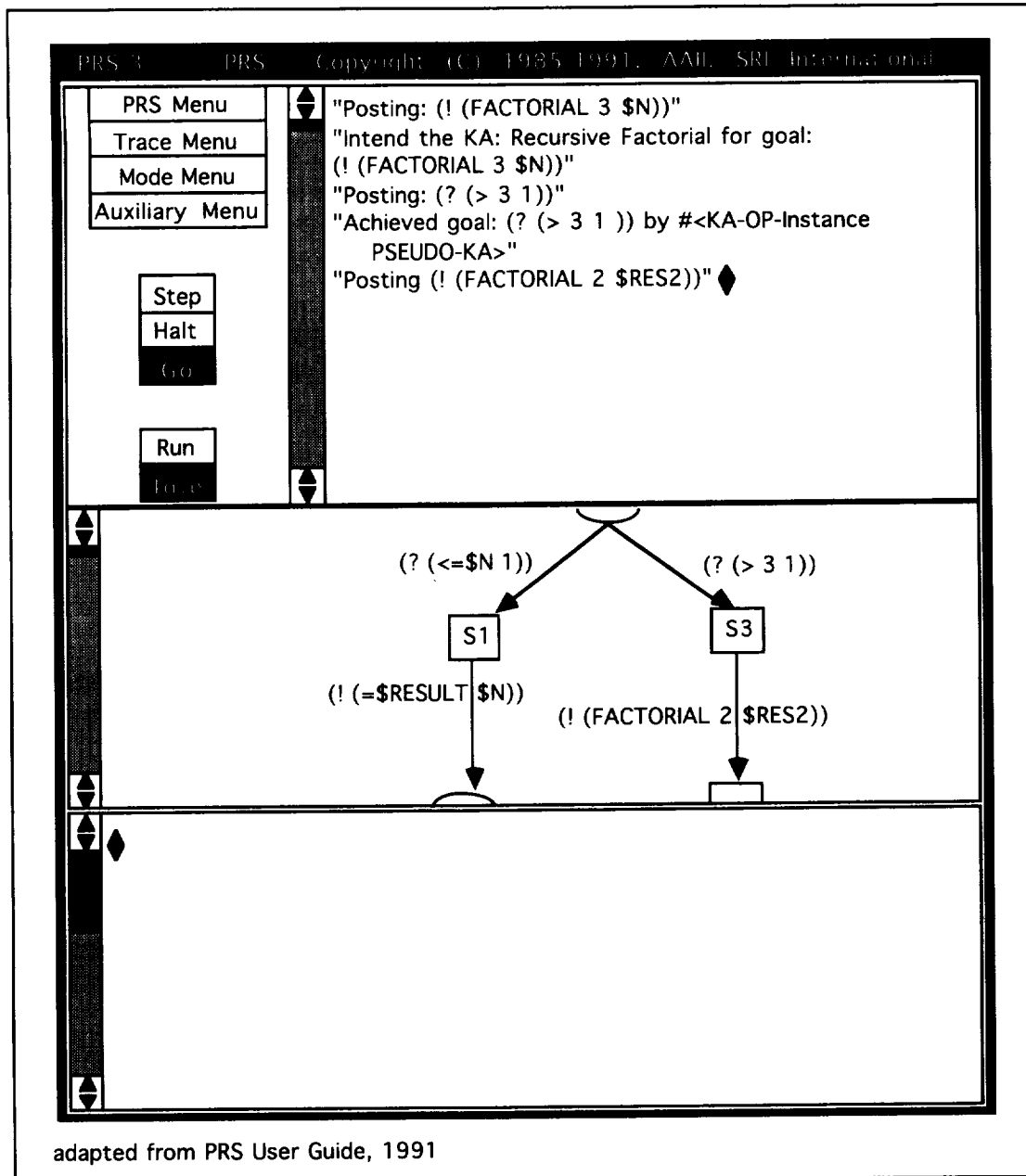


Figure 2-6. Examples of the IO Window, Text Trace Window, and Command Window.

The Text Trace Window is located in the middle of the workspace. This region provides information for monitoring the execution of plans. Information about plan execution can be viewed in both a message format and a graphics format. A scrolling message window displays messages describing the plan step being monitored (i.e., what the intelligent system is doing) and the goal to be achieved (i.e., why this plan is being executed). These messages are listed in the order of occurrence. A scrolling graphics window displays a graphical version of the plan steps being executed. Since the graphic is usually larger than the window viewport, the view automatically shifts to show the portion of this graph being monitored. Both the message window and graphics window can be viewed simultaneously. See figure 2-6 for an example of the Text Trace Window.

The graphic of plans in the Text Trace Window is an example of a representation shared by the operator and the PRS instance (see also section 2.3). This graphic is monitored by the operator as plans are executed. It illustrates the monitored process situation as it develops and shows how the intelligent system responds to that situation. There are some challenges in using this graphic operationally, however. There is no integrated view of plans across PRS instances. Large procedures can be difficult to follow, since the entire procedure cannot be displayed at one time. Additionally, the abrupt movement through the graphic as the procedure is executed can be difficult to track visually.

The Command Window is located at the bottom of the workspace. This scrolling text window is used to enter commands to PRS. A variety of types of commands can be issued. To specify the desired type of command, a command mode is selected from the Control Window. The following command modes are provided:

- *default*: enter LISP command
- *message*: send message to PRS instance
- *consult*: request all facts matching expression entered
- *fact*: assert a fact
- *goal*: establish a goal
- *relevant KA*: request all KAs responding to the fact or goal entered

PRS automatically logs the messages displayed in the IO Window, the Text Trace Window, and the Command Window. These messages are not accessible from the screen, but can be saved to a file for use after operations are complete.

As mentioned in section 2.3, information from the multiple instances of PRS has not been integrated for use by a flight controller during operations. User interface issues associated with supervising multiple instances of PRS include (1) integrating the displays associated with these instances into a single workspace, (2) navigating through this workspace, (3) focusing operator attention on instances and information describing the situation, and shifting attention when the situation changes, (4) illustrating information relationships that span instances, (5) tracking which instance is being viewed, and (6) effectively managing real estate usage (such as displaying graphics of KAs that are larger than the window viewport).

2.5 Design Process

The PRS was developed to investigate artificial intelligence (AI) applications for Space Station Freedom. This system was developed by SRI International (M. Georgeff⁸, PI and F. Ingrand) and funded as an AI RTOP by the Office of Aeronautics and Space Technology (OAST). Initial development of the system was done at the Ames Research Center. Within the last year, the software has been transferred to Johnson Space Center (JSC) for operational evaluation under the Real-Time Data System (RTDS) project. PRS differs from most RTDS systems because it does not use rule-based technology and does not use the RTDS data acquisition software.

This system was developed as a proof-of-concept prototype for the Space Station Freedom Program. The user community for this system consists of Space Shuttle flight controllers (M. Barry/RSOC). These users have played an active role in system development. They have served as the domain experts for the RCS. This was possible because similar RCSs are used in both

⁸ M. Georgeff formerly worked for SRI International. He is currently at the Australian AI Institute, where he continues to work on the PRS project.

programs. They have decomposed the selected malfunction procedures into reusable "pieces" to derive the KAs. Instead of holding traditional knowledge acquisition sessions with the software developers, these flight controllers used the Knowledge Editor in PRS directly to represent procedural information in electronic form. They also used the Structure Editor to represent the structure of the monitored process⁹. They are currently evaluating the system in representative operational scenarios using data from the Single System Trainer.

2.6 Study Method

Information about PRS was obtained from demonstrations and interviews with Matt Barry in March (6, 20), April (23, 28), and August (28) of 1992. The case data sources cited in section 2.7 were also used, and the figures shown throughout section 2 were derived from figures included in these case data sources.

Project Representatives

- Matt Barry (Rockwell Space Operations Company, RSOC)

2.7 Case Data Sources

Georgeff, M., and A. Lansky. *A System for Reasoning in Dynamic Domains: Fault Diagnosis on the Space Shuttle*. SRI International, Technical Note 375. January 1986.

Georgeff, M., and F. Ingrand. Decision-Making in an Embedded Reasoning System. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Detroit, Michigan. 1989.

Georgeff, M., and F. Ingrand. Monitoring and Control of Spacecraft Systems Using Procedural Reasoning. *Proceedings of the Space Operations, Automation, and Robotics Workshop*. Houston, TX. July 1989.

Georgeff, M., and F. Ingrand. Real-Time Reasoning: The Monitoring and Control of Spacecraft Systems. *Proceedings of the IEEE Sixth Conference on AI Applications*. Santa Barbara, CA. March 1990.

Georgeff, M., and F. Ingrand. *Research on Procedural Reasoning Systems, Final Report - Phase 2*. SRI International. March 1990.

Ingrand, F., M. Georgeff, and A. Rao. Embedded Diagnostic Systems, submitted to *IEEE Expert*.

⁹ This approach also enables quick upgrades of the knowledge base when the system configuration changes.

SECTION 3

COOPERATING EXPERT SYSTEM

3.1 System Description

The CoopES is a distributed system developed to demonstrate cooperation among multiple expert systems performing real-time fault management. CoopES consists of two expert systems for managing faults in different systems of a space vehicle, and the software necessary for these systems to communicate and coordinate their activities. Both expert systems were developed previously as stand-alone systems for detecting, diagnosing, and recovering from failures in the Space Shuttle EPS and the RCS, respectively. Simple simulations of the Space Shuttle RCS and EPS are built into each expert system to provide data for demonstration and testing.

Unlike most systems in the case study, CoopES was designed to operate without significant user involvement. CoopES was included in this study to investigate how interaction among distributed software agents provides insights into interaction among humans and such software agents. In effect, the human is being considered as another agent in a distributed software system.

CoopES differs from other cases in another way. Most systems in the case study are planned for use by flight controllers during space operations. CoopES was never intended for operational use, and was built as a proof-of-concept prototype. The technical challenge for CoopES was to demonstrate reliable communication and cooperation among distributed expert systems developed for stand-alone use.

Both the RCS and EPS expert systems are rule-based systems developed using SCLIPS. The user interface to the RCS expert system was implemented using TAE and the user interface to the EPS expert system was implemented using Sunviews. The remainder of the software was developed using SCLIPS, C, and Unix, with one exception. Software for coordinating these systems (called the System Monitor) is a frame-based system built in LISP with a user interface built in C and the XWindows system and communications software built using LISplink. The CoopES software runs on multiple Unix workstations (typically Sun workstations) networked together.

3.2 Intelligent System and Functions

The CoopES architecture is a multi-agent architecture, where agents are software processes that reason about local and global facts and plans, and that communicate with other processes about these facts and plans. Agents are organized hierarchically, with a supervisory agent coordinating the activities of two fault management agents. These subordinate agents pass status information and local plan requests to the supervisory agent, which can approve these plans or change them as needed to form a global plan. Other agents are provided to monitor agent health, to configure agents, and to establish agent communication. For the purposes of demonstration, an agent is also provided for introducing faults into the space vehicle system. These agents communicate by passing messages. A blackboard is used for communication between the fault management agents and their associated simulations. CoopES consists of the following agents:

- **System Monitor**

The System Monitor (SM) agent coordinates the activities of the RCS and EPS expert systems. It receives plans selected by these two intelligent agents, coordinates the plans received, and monitors the execution of each step in the overall plan. The user interface to the SM, the SMInterface, receives messages from the SM about plan execution and displays a graphical representation of those plans.

- **Reaction Control System**
The RCS expert system (called PropSys) manages faults in the RCS subsystem of the Space Shuttle propulsion system. It is a rule-based system with approximately 300 rules. The RCSOI is the user interface for graphically displaying information from this expert system. A simple simulation of the RCS is embedded in the expert system for use in demonstration and testing.
- **Electrical Power System**
The EPS expert system (called EpSys) detects faults in the Space Shuttle electrical power system. It is a rule-based system with approximately 200 rules. The EPSOI is the user interface for graphically displaying information from this expert system. A simple simulation of the EPS is embedded in the expert system for use in demonstration and testing.
- **Heartbeat**
This agent monitors all other agent processes running in the system, and restarts any agent that ceases functioning or loses the ability to communicate.
- **Murphy**
This agent is used to introduce monitored process faults into a scenario-based demonstration of CoopES.
- **Server**
The Server agent uses a preset configuration file to determine the allocation of agent processes to machines, launch these agents, establish appropriate system environments, and establish agent-to-agent communication.
- **Configuration Manager**
The Configuration Manager (CM) agent provides the ability to configure, launch, and control the different hardware and software components of CoopES demonstrations. This agent includes a graphical user interface.

The typical agent configuration for CoopES is shown in figure 3-1.

The CM is used for graphically building an agent configuration and to launch the agent processes specified in this configuration. Configurations pre-defined as demonstration scenarios include the following:

- Reactive Planning (no agent cooperation)
- Sequential Diagnostic Action (hierarchy of agents with sequential events)
- Overlapping Diagnostic Action (hierarchy of agents with overlapping events)

The Server starts each agent process specified in the configuration and provides each agent with default start-up data. It then starts the Murphy agent, which introduces faults into the monitored process simulations that correspond to the selected scenario. At this point, the Server has completed its activities.

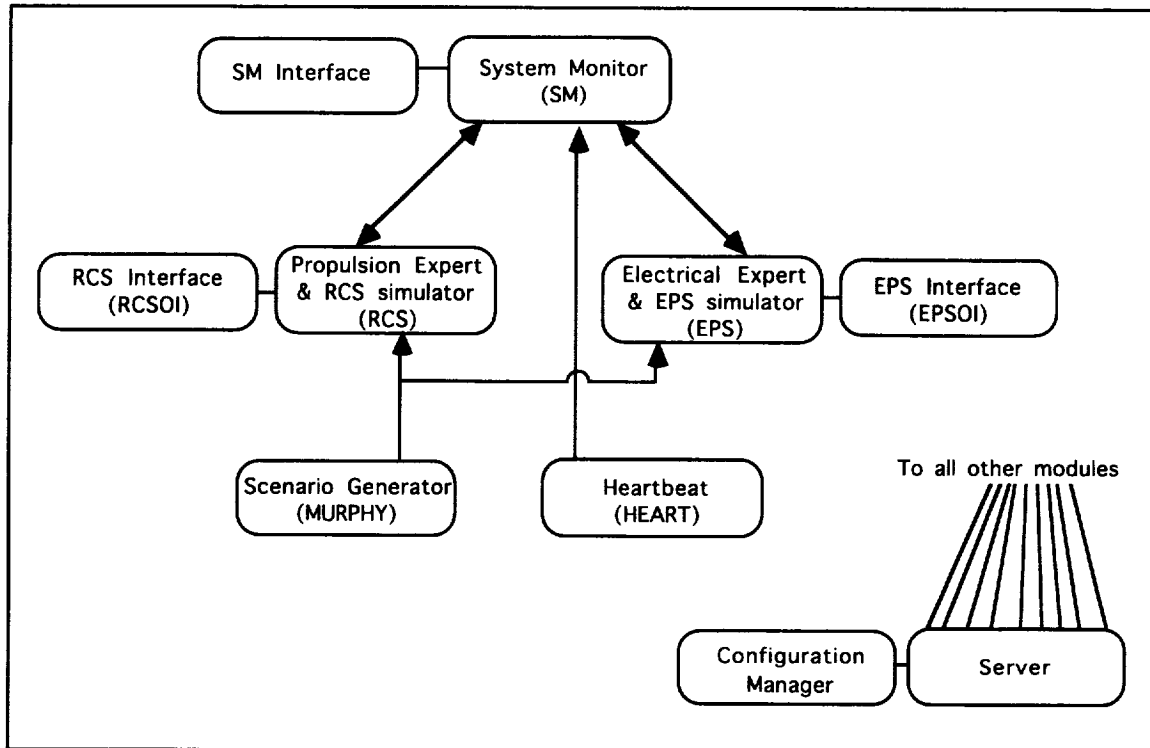


Figure 3-1. Typical CoopES Configuration.

Once the agent processes have been configured and launched, the PropSys and the EpSys begin to monitor simulated RCS and EPS parameters, respectively. As problems are introduced into the simulated RCS and EPS using the Murphy software, these fault management agents detect problems in their respective domains¹⁰. When a problem is detected, the fault management agent selects a local recovery plan from a pre-defined set of plans (scripts)¹¹. These local plans (i.e., plans intended for execution) are merged by the SM into a global plan agenda. Each fault management agent executes the plan steps relevant to its monitored process.

Throughout operations, the Heartbeat agent manages the health of all agent processes. Health monitoring consists of ascertaining that all agents are still active by periodically requesting a response from them. If an agent fails to respond, Heartbeat assumes that the agent is no longer "alive," and starts up a new agent on an available platform.

Synchronizing the information communicated among distributed agents is an important issue for CoopES. Concurrent agent processing and network transmission delays can result in stale information being provided to an agent, or in information sequences that are received out of order (obsolete or inconsistent information). These information synchronization problems are solved in CoopES by limiting agent communication to message passing through the supervisory agent (the SM) and by using communication protocols. Synchronization between PropSys, EpSys, and their associated simulations was not changed from their original designs, which embedded the simulation in the expert system agent with facts shared between them¹².

¹⁰ The detection of faults by expert systems is based on fixed thresholds defining nominal operating conditions.

¹¹ All CoopES plans are static and are derived from the malfunction procedures for the Space Shuttle RCS and EPS.

¹² This design eliminates data acquisition problems, such as time synchronizing data, but makes agents less flexible (can't change data sources easily).

3.3 Human-Intelligent System Interaction Functions

Because CoopES was designed for autonomous operation, most of the human interaction with the system occurs prior to starting it. Typical human tasks include building fault scenarios and configuring the system. At run time, human involvement is limited to observing the intelligent software agents while operating. Information from three different agents is monitored during system operations:

- SM: status and activities of agents executing the plan
- PropSys: state and status of RCS space system
- EpSys: state and status of EPS space system

Information from these distributed agents has not been integrated to assist the human in monitoring fault situations (i.e., three screens of information are provided on three different hardware platforms).

Although CoopES is designed for limited interaction with humans, many observations and issues concerning interaction among intelligent software agents are relevant to interaction among humans and software agents. The types of information needed and activities performed by the supervisory agents in CoopES (specifically the SM and Heartbeat) are very similar to the information requirements and activity descriptions for humans supervising intelligent systems (Malin and Schreckenghost, 1992). Issues related to supervising intelligent software agents are discussed in the remainder of this section, particularly (1) coordinating agents using plans, (2) solving agent communication problems, and (3) managing agent errors, including assessing agent (and overall system) health.

The supervisory agent in CoopES (SM) coordinates subordinate agents (PropSys and EpSys) by managing the activities that these agents perform (i.e., managing agent plans). Each subordinate agent proposes a local plan based on its assessment of the failure situation. Before executing these plans, the supervisory agent identifies and resolves global issues in local plans. Specific activities include identifying resource conflicts and dependencies in these plans (e.g., power bus must be good for RCS leak recovery), and selecting different plans that accommodate these dependencies. To modify and combine local plans into a global plan agenda, the supervisory agent needs information about the vehicle system monitored by the subordinate agent (e.g., health, availability, configuration, redundant capability), fault diagnoses by the subordinate agents, and agent status (from Heartbeat). As part of coordinating activities, the supervisory agent monitors the execution of the plan steps by the subordinate agents to verify that these steps are completed. If a plan fails to complete, performance of the multi-agent system can be impacted.

The ability to coordinate agents and execute plans can be impaired by communication failures. The CoopES project identified a number of communication failures caused by problems in synchronizing or transmitting information (see section 3.2). These failures can result in agent errors, where the receiving agent misses information, or reasons with information that is obsolete, inconsistent, or out of order. For example, in one demonstration scenario an incorrect plan was selected by the SM based on inaccurate information due to a communication failure¹³. A common response to communication failure is re-transmitting information. With asynchronous communication, this can result in situations where the transmitting agent repeats the transmission unnecessarily (i.e., doesn't wait a sufficient amount of time for a response indicating the transmission was received). Problems due to repetitive messages can impact performance of the multi-agent system. For example, in one scenario the PropSys and the SM engaged in a meaningless dialogue (called a

¹³ SM did not wait for activities altering system state to complete, and state information to stabilize, before using the information to select a plan

Navajo dialogue¹⁴) in which a fault was repeatedly diagnosed, bringing CoopES effectively to a halt. Repetitive messages in CoopES are handled by modifying agent reasoning to prevent them from being issued. A related problem occurs when an erroneous agent floods another agent with information (called a runaway agent). This problem is controlled in CoopES by limiting communication via CPU allocation (i.e., a decoupling coefficient limits how much CPU is allocated to communication).

Most agent errors caused by communication failure are managed in CoopES by designing agents to prevent errors from occurring (e.g., do not repeat messages) and, when they do occur, to have limited impact (e.g., limit communication from runaway agent). Agent errors resulting in total loss of communication, however, are detected and managed in real time by the Heartbeat agent. Loss of communication is detected by periodically forcing each agent to verify it can communicate (essentially a low-level assessment of the health of individual agents). Performance problems (like the meaningless communication of the Navajo dialogue) have been observed in CoopES, but are not detected and managed in real time. Methods for evaluating both individual agent health and overall system performance (e.g., detect Navajo dialogue) are required to assess the health of multi-agent systems.

Managing agent errors includes recovering from those errors. Heartbeat responds to a loss of agent communication by reallocating the agent's tasks to another agent. It assumes that the erroneous agent is not repairable, and creates a new agent process. Reassigning an agent's tasks includes establishing communication between the new agent and other agents in the system, and starting the new agent from a default configuration. Such task reassignment would be more effective if the new agent could be given some understanding of ongoing fault situations (e.g., knowledge of previous events provides context for interpreting current events).

3.4 Supporting User Interface Capabilities

As discussed in the previous section, the human plays a limited role in CoopES. User interface displays are only provided for the purpose of demonstrating autonomous operations. The CoopES user interface consists of three operational displays and one system configuration display, corresponding to one display per intelligent software agent¹⁵. The workspace of CoopES consists of the following displays shown concurrently on four workstations:

- **SMInterface:** displays agent activities as they occur and the status of agents executing plan steps
- **EPS interface (EPSOI):** displays state and status of the EPS
- **RCS interface (RCSOI):** displays state and status of the RCS
- **CM:** provides capability to configure and start up software agents

Each display has a unique format. In fact, the original user interfaces developed for the stand-alone expert systems have been used virtually without change. These separate displays have not been integrated into a single user interface because the system was not designed for use by a

¹⁴ The Navajo dialogue is a communication failure where multiple agents become trapped in a repeating cycle of purposeless communication and action (similar to an endless loop in traditional software). The name of this failure refers to the pattern observed on the user interface during such interaction, which resembles the pattern on a Navajo blanket. In this example, a fault was diagnosed and a plan modification was suggested, but could not be completed, before the fault was re-diagnosed.

¹⁵ Heartbeat is the only exception to this. Information from Heartbeat is displayed on SMInterface.

human (except to set up and start system). Figure 3-2 illustrates the CoopES multi-screen workspace. Each of these displays is described below.

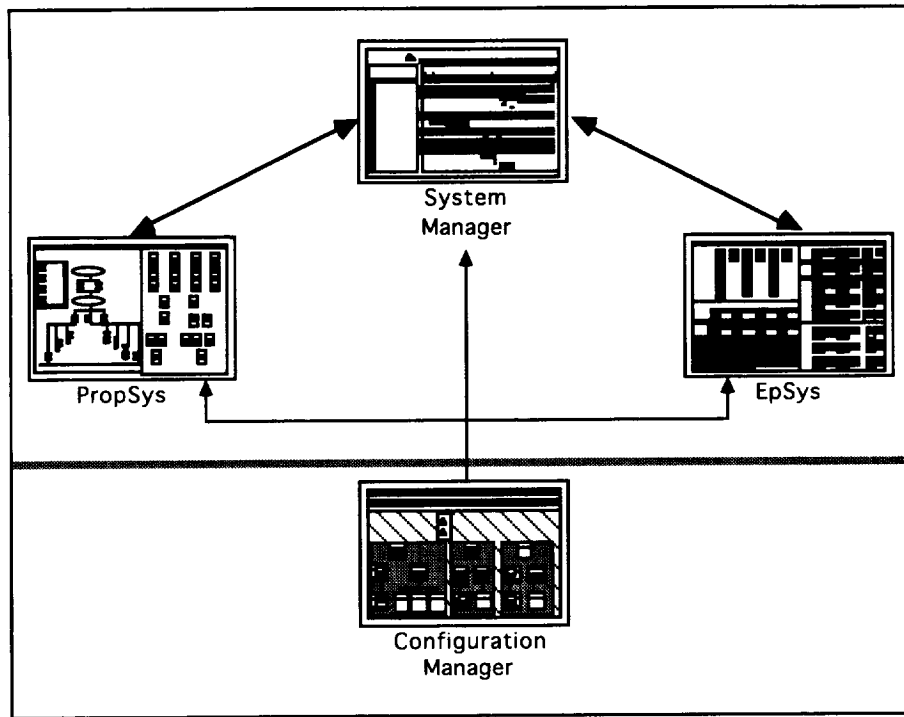


Figure 3-2. The Workspace of CoopES.

The SMInterface is the only display in the case study that was designed for the purpose of managing intelligent software agents. It provides information for monitoring agent activities and plan execution. It illustrates activities performed by all active agents concurrently (using a timeline) and shows plan steps as they are executed. These activities include supervisory activities performed to manage agents (e.g., reallocate tasks to new agent). It also provides a health assessment of all agents. Patterns of activity on the timeline can indicate system performance problems (e.g., Navajo dialogue is characterized by a repeating pattern of interaction). Agent communication is illustrated by explicitly marking messages as events on the timeline. A history of agent status and activities can be reviewed by scrolling through the timeline or activity list. Although the SMInterface can indirectly provide information about the fault situation (e.g., anomaly occurred if the agent is performing fault management activities), its primary purpose is to illustrate information for monitoring the intelligent agents.

The SMInterface shows information from the SM and Heartbeat concerning the status and activities of agents executing the global plan. The left side of the display lists labels for the graphic timeline items on the right side. These labels are a combination of agent health and status assessments (e.g., RCS-IS-ALIVE, indicating that the Heartbeat agent believes the PropSys agent is functioning properly) and agent activities corresponding to the steps of the global plan (e.g., RCS-DETECTION, indicating that the PropSys agent has detected an anomaly). Labels are added dynamically to this status and activity list as situations develop. Ongoing activities are indicated by timeline bars, and single events (e.g., message sent) by triangular markers. Current time is marked by a hollow diamond in the first line of the timeline. See figure 3-3 for an example illustrating the SMInterface display.

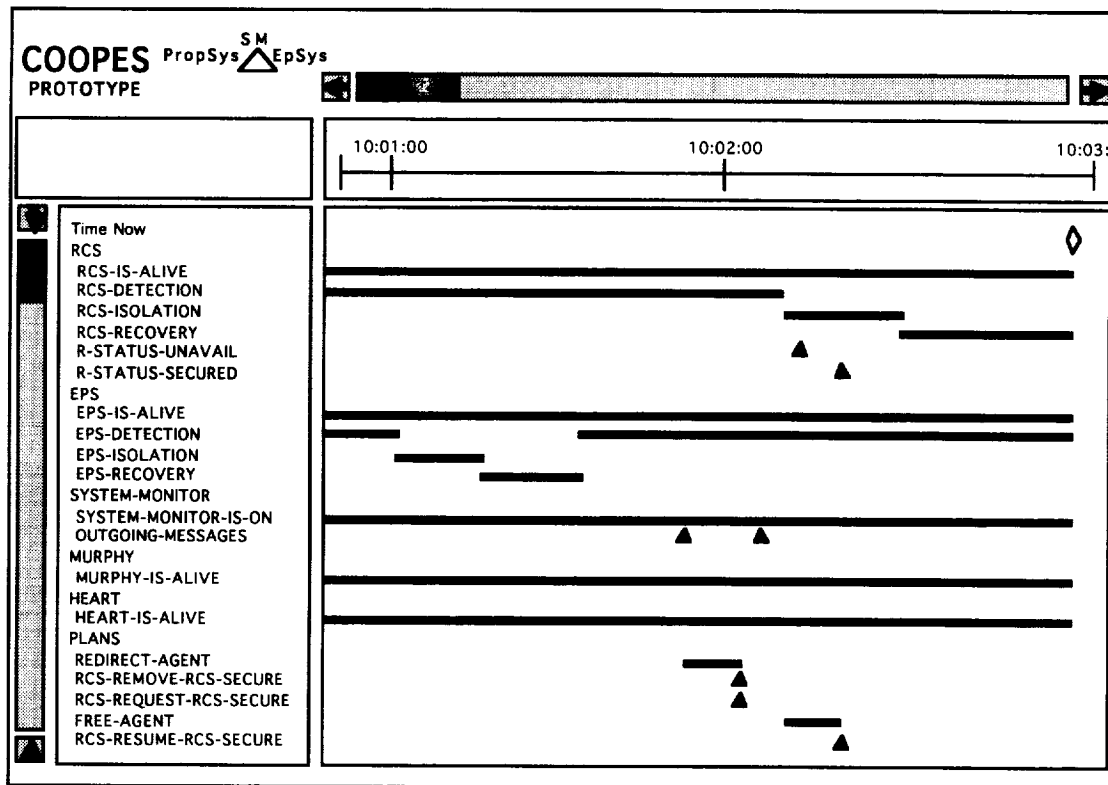


Figure 3-3. Example of the SM Interface Display.

The EPSOI displays fault diagnosis information from the EpSys agent. This display consists of a scrolling message window and multiple digital panels. These panels are grouped by the subsystems in the EPS (i.e., fuel cells, power distribution and control, and power reactant storage and distribution). The status of parameters within these subsystems is displayed in the digital panels. Diagnostic conclusions are displayed in a scrolling message list. User control buttons are provided for setting up and running EpSys. These buttons are not used during the CoopES demonstrations, and are an artifact of the original development as a stand-alone system. See figure 3-4 for an example of the EPSOI display.

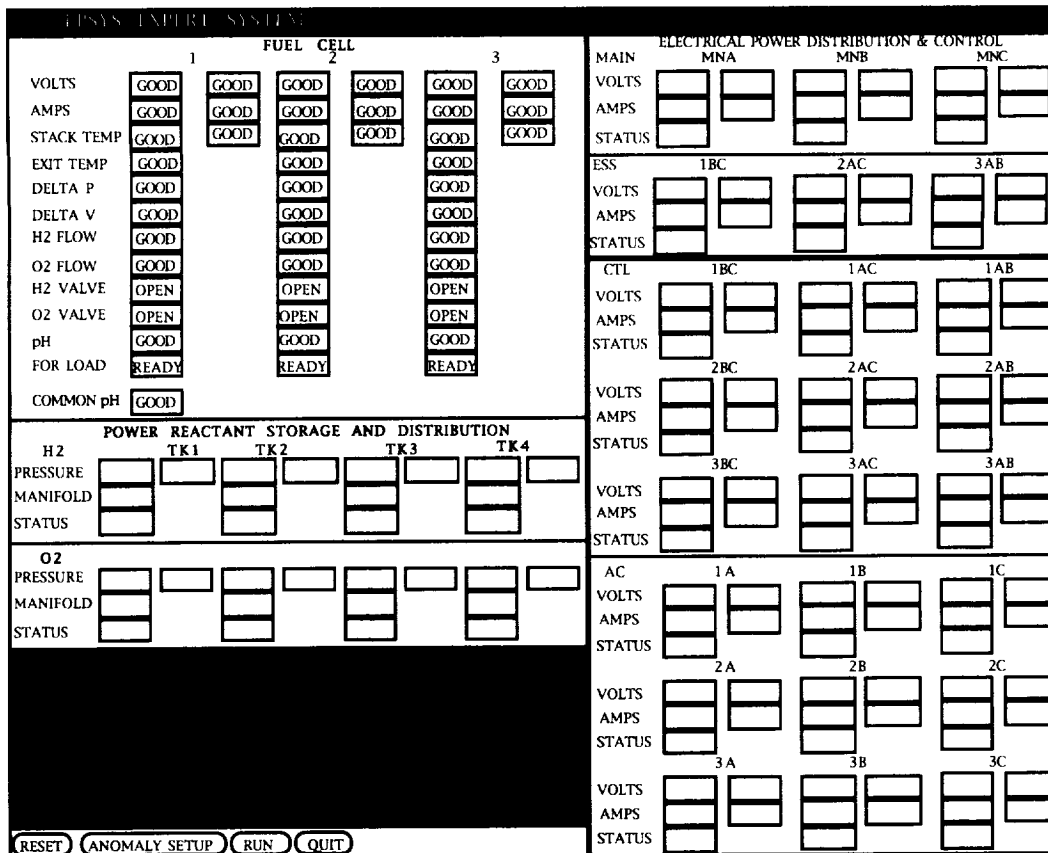


Figure 3-4. Example of the EPSOI Display.

The RCSOI displays fault diagnosis information from the PropSys agent and parameters from the simulation of the RCS. The RCSOI display has two parts. The left side of the display shows a high-level schematic of either the left or right aft RCS¹⁶. This schematic illustrates the path of fuel flow through the components of the RCS. The valve components controlling this flow are shown as dynamic icons whose shape and color to illustrate the combined state and status of the valve. A legend defining possible valve states and statuses is shown in the upper left corner of the display. The right side of the display provides sensor measurements and intelligent system conclusions about the RCS system shown in the schematic. The current values of data from the simulator (e.g., tank temperatures, pressures) are displayed on digital panels. Intelligent system assessments of the state (e.g., open, closed) and status (e.g., good, failed) of the valves in the system are also displayed on digital panels. The RCSOI provides no user control capability. See figure 3-5 for an example of the RCSOI display.

¹⁶ The aft RCS consists of duplicate propulsion systems, one each on the left and right sides of the vehicle.

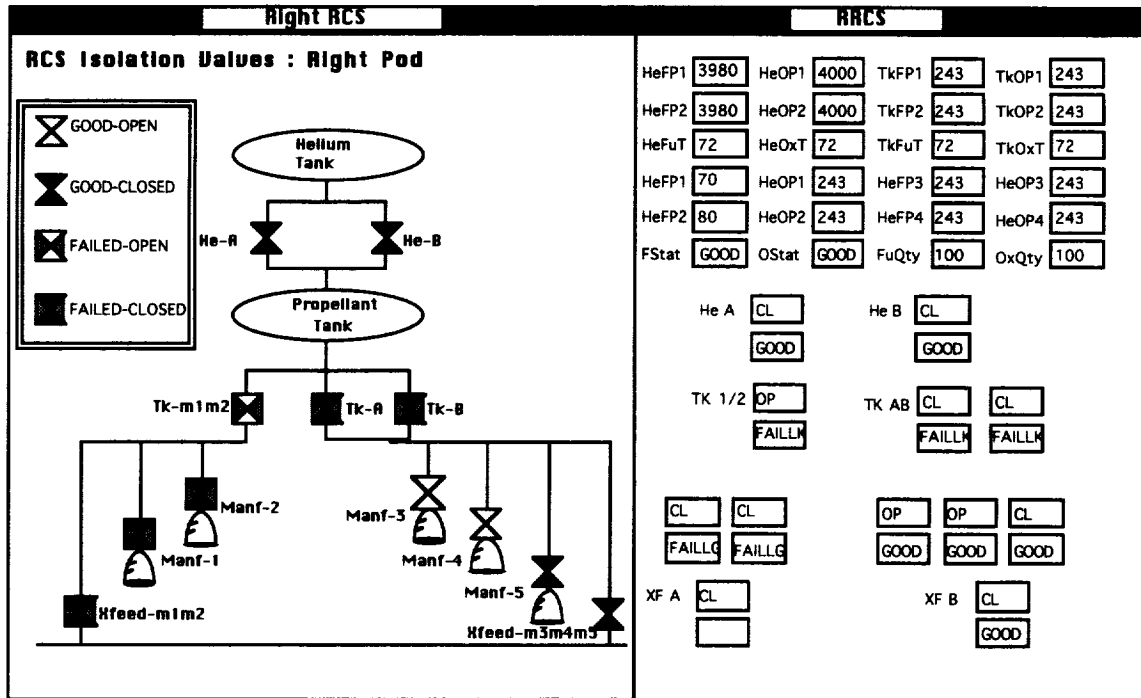


Figure 3-5. Example of the RCSOI Display.

The CM display is used to configure CoopES and start the system demonstration. Configuring CoopES consists of allocating agent processes to the available hardware platforms and specifying a demonstration scenario. As illustrated in figure 3-6, the CM display consists of a pull-down menu and a graphic display of agent configuration. An agent is allocated to a host by selecting a node¹⁷ icon on the graphic and specifying a process and a hardware platform. Using the menu, the user can launch and halt CoopES, launch and reset agent processes, and select pre-defined fault scenarios. The graphic of system configuration illustrates all active agents and their host platforms. This display is not used after starting the system.

3.5 Design Process

CoopES is a proof-of-concept prototype developed to demonstrate the feasibility of distributed cooperating expert systems, and to define requirements for building such systems. It had a further constraint to reuse stand-alone expert systems and integrate them into the multi-agent architecture. Many difficulties were encountered in reusing stand-alone expert systems. These difficulties resulted from incompatibilities in system architectures, knowledge representations, and communication approaches.

CoopES was built by McDonnell Douglas Space Systems Company (MDSSC) (J. Rufat, C. Clark, G. Watts) for the Software Technology Branch (C. Culbert/PT4). It was developed iteratively by a large development team. The development team was partitioned into groups developing specific software agents. Users were not actively involved in this development, and domain knowledge was derived from documentation (e.g., Space Shuttle malfunction procedures). The CoopES project has been completed, and the results of the project have been documented (MDSSC, 1990).

¹⁷ A node consists of a host platform and the agent processes running on that platform.

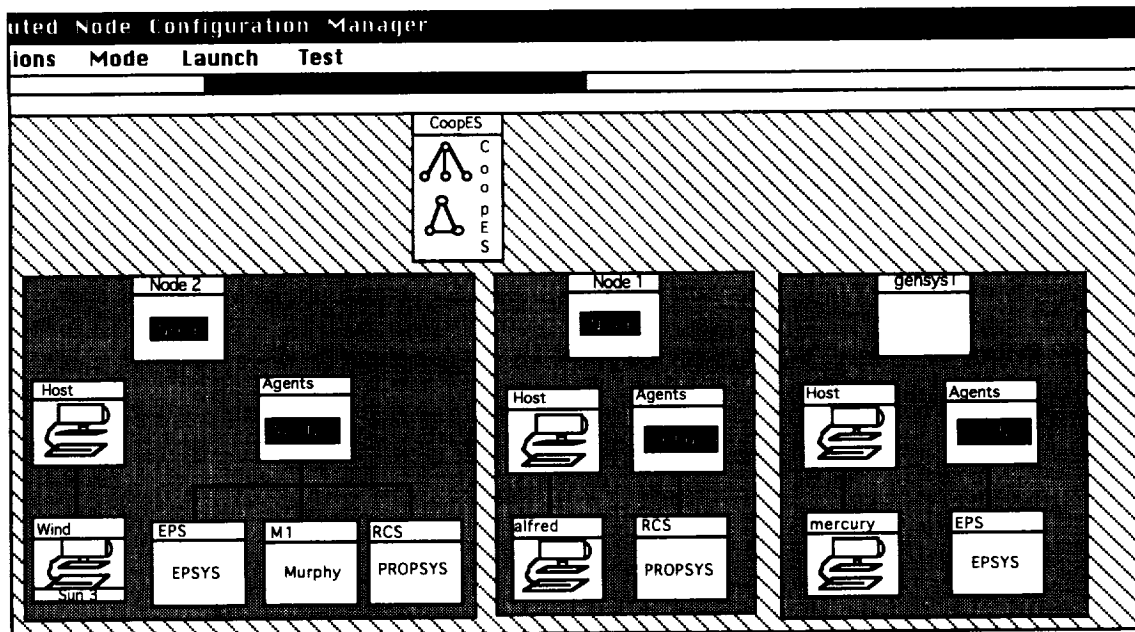


Figure 3-6. Example of the Configuration Manager Display.

3.6 Study Method

Information about CoopES was obtained from a demonstration on January 28, 1992, from interviews (1/28/92) with G. Watts and C. Clark on January 28 and February 27 of 1992, and a phone conversation with J. Rufat on July 6, 1992. The case data sources cited below were also used. The figures used in this section were derived from figures included in the documents listed in section 3.7.

Project Representatives

- Chris Culbert (Software Technology Branch, NASA/JSC)
- Colin Clark (MDSSC)
- Jorge Rufat (MDSSC)
- Grayum Watts (MDSSC)

3.7 Case Data Sources

Benson, Sara. *EPSYS Project Overview*. Operations Technology Initiative Task 5. McDonnell Douglas Space Systems Company. December 1988.

Bochsler, Dan. *Autonomous Space Vehicle Operations Simulation Testbed (AUTOPS)*. Briefing in February 1990.

Culbert, C., J. Rufat, G. Watts, P. Vu. *Cooperating Expert Systems*. Briefing by Software Technology Branch. December 1990.

Malin, J. and Schreckenghost, D. *Making Intelligent Systems Team Players: Overview for Designers*. NASA Technical Memorandum 104751. June 1992.

MDSSC. *Cooperating Expert System Lessons Learned (Draft)*. November 1989.

MDSSC. *Operations Technology Initiative OTI Task 5 - On-Board Systems Management..* FY 90 Report - FY 91 Project Plan (DRAFT). April 1990.

Rufat, J. *Cooperating Expert Systems for Autonomous Spacecraft Operations Assumptions and Design Guidelines (DRAFT)*. McDonnell Douglas Space Systems Company. March 1990.

Rufat, J., G. Watts, O. Carey, W. Parrott. *Distributed Cooperative Systems for Advanced Automation: Tradeoffs*.

SECTION 4

EXTENDED REAL-TIME FEAT

4.1 System Description

The Extended Real-Time FEAT (ERF) system is a fault management tool for use by ground flight controllers in the Combined Control Center (CCC)¹⁸. The objective of the ERF project is to extend the capabilities of FEAT by providing automated real-time analysis of failure models using telemetry data for managing faults in a vehicle system (i.e., the monitored process). ERF assists the flight controller in determining failure causes (fault isolation), failure propagation paths, and failure effects (or impact assessments). Unlike most systems in the case study, ERF is not custom-built for a specific vehicle system, but is instead a generic tool for use by all flight controllers managing such systems. To use this tool, a system-specific failure model must be built, or a FEAT failure model can be imported from vehicle system designers. A feasibility demonstration of this system is being developed for the Space Station Freedom propulsion system.

As baseline software for the CCC, ERF complies with the CCC software and hardware standards. It is being built to run on Sun and DEC workstations in a Posix environment and is being written in the Ada language. Displays and controls follow human-computer interface standards (e.g., XWindows standards) and guidelines of the program.

4.2 Intelligent System and Functions

The ERF analyzes a model of possible failures in the monitored process to determine the set of faults suspected as causing the observed onboard system failure conditions, including identifying the common root cause for multiple failure conditions. To perform these functions, the off-line FEAT¹⁹ has been modified for use during real-time operations. ERF automatically analyzes the failure model in a way similar to the manual analysis a flight controller would perform. The failure model is represented as a directed graph (digraph), where nodes representing failure sources (circles) are connected by edges (arrows) indicating causal relationships (arrows point from cause to effect). A significant feature of ERF is that it extends the FEAT digraph model to include sensor nodes associated with telemetry data and caution and warning (C&W) messages. Figure 4-1 illustrates a simple digraph of a light bulb connected to a power source via a switch. Failure of the bulb to light can be caused by (1) a bad bulb, (2) a bad switch, or (3) a power problem (ground or power source failure). For complex space systems, failure digraphs are considerably more complex than the digraph in this example, often containing hundreds of nodes.

ERF performs two types of functions, identification of the root cause of a failure and prediction of the consequences (or impacts) of a failure. Failure causes are identified by eliminating failure nodes from the digraph that data indicate are not failed or that the model indicates could not be a root cause. Failure consequences are predicted by propagating failures to nodes downstream²⁰

¹⁸ The CCC will be used for ground-based support of the Space Station, and eventually for the Space Shuttle.

¹⁹ FEAT is a tool developed by Lockheed Engineering and Sciences Company (LESC) for NASA's Intelligent System Branch (ER2). FEAT can be used by system designers for failure modes and effects analysis (FMEA). FEAT is available for both a Macintosh and a Unix platform.

²⁰ Nodes *downstream* from Node A are nodes connected to paths coming out of Node A (i.e., to the right of Node A, assuming that arrows point to the right). Similarly, nodes *upstream* from Node A are connected to paths coming into Node A (i.e., to the left of Node A).

from the cause of the failure. These two functions are described in the remainder of this section. The identification of root causes of a failure is discussed first. The prediction of failure consequences is discussed at the end of this section.

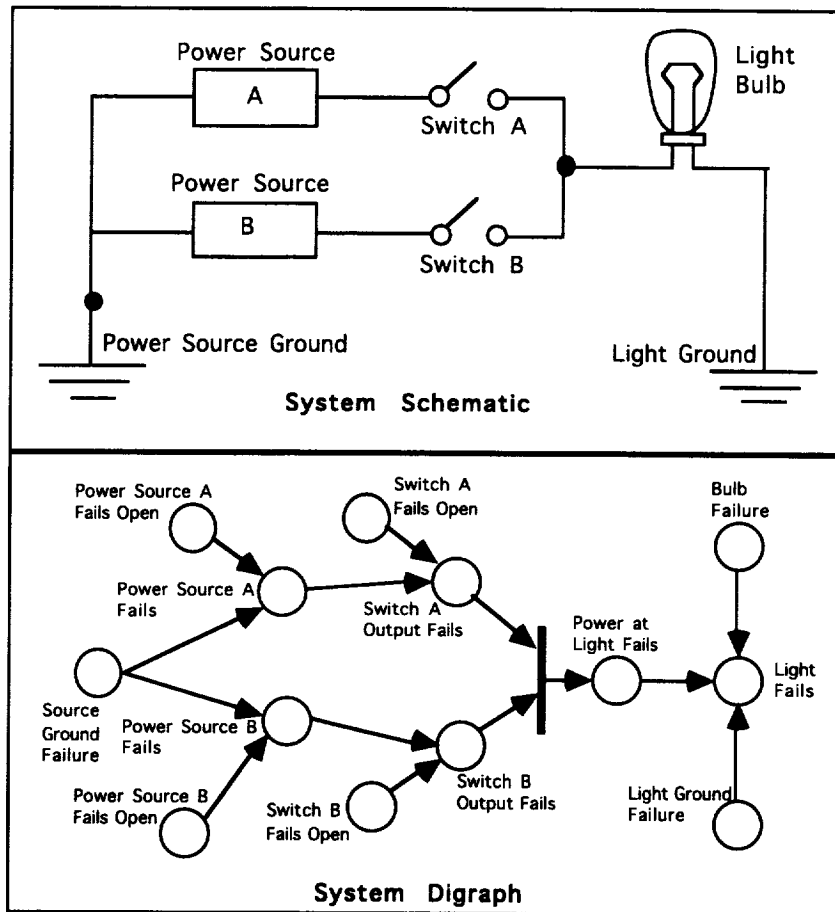


Figure 4-1. A System Schematic and its System Digraph Model (Schier and Gantzer, 1991).

ERF identifies the minimum set of potential root causes of a failure (called *faults*) by eliminating nodes from the failure digraph. ERF analysis includes identifying a common cause of multiple failures (i.e., cascading alarms caused by common root fault) and excluding secondary failures from the list of potential failure causes. To do this, ERF interprets data from the sensor nodes in the model, constructing a list of potential failure sources based on this data. ERF eliminates potential failure sources corresponding to inconsistent or impossible failures. The remaining nodes in the digraph represent the minimum set of potential faults. These steps are discussed in greater detail below.

ERF categorizes sensor nodes according to the telemetry data and C&W messages received for each sensor node. Data are compared to nominal operating limits²¹, and data quality and availability is checked. The resulting sensor categories are listed below:

- *observable* - sensors in the failure model that are on the telemetry object list (TOL), i.e., data are available
- *good* - sensors with data of good quality that are within nominal operating limits, i.e., good data indicates no failure
- *bad* - sensors with data of good quality that are outside of nominal operating limits, i.e., good data indicates a failure
- *unknown* - sensors in the model that are not on the TOL, i.e., data are unavailable, or sensors in the model that are on the TOL but have unacceptable data quality, i.e., data of bad quality

ERF performs a series of automated calls to the embedded FEAT system emulating the actions a human flight controller would take to identify the minimum set of suspected faults. ERF first constructs another digraph representing the current condition of all observable sensor nodes. Good and unknown sensor nodes are grouped and eliminated from this new digraph, since they cannot assist in identifying the failure cause. Next, a list of the potential failure sources is constructed by identifying all the singletons and doubletons²² that would have to occur to reach the root node of this new digraph (using the FEAT target function). This list is further pared by eliminating the following nodes from the digraph:

- all single nodes upstream from a known good node
- all artificial nodes that are artifacts of building the model
- all nodes in doubletons with a partner node that was removed in a previous step

The portion of the digraph that remains after these steps represents the set of candidate nodes for the root cause of the failure. If at any point in this pruning process a single node remains in the digraph, it is assumed to be the root failure node.

The second type of functionality provided by ERF is prediction of failure consequences. A failure impacts all nodes directly in the downstream path. When doubletons are in the downstream path, all nodes downstream of the doubleton may be impacted also. Thus, ERF determines the set of all possible failure consequences. The operator is responsible for interpreting these consequences (e.g., identify the next worst failure). Since ERF uses more conservative operating limits than the C&W system onboard the vehicle, it can be used to predict expected C&W messages and failure impacts before they actually occur.

4.3 Human-Intelligent System Interaction Functions

ERF supports the flight controller in detecting faults in the monitored process, isolating their cause, and assessing the impacts of faults. ERF does not support monitoring nominal operations, and only initiates activity upon receiving a C&W message (and associated out-of-limits telemetry data

²¹ The nominal operating limits used by ERF to detect alarm conditions are more conservative than the operating limits used to detect alarm conditions onboard the vehicle. This permits detecting and responding to potential alarm conditions before they are annunciated onboard the vehicle.

²² A *doubleton* is a member of a pair of failure nodes both of which must fail before nodes downstream from the pair will fail. A single failure relative to a node downstream is known as a *singleton*.

and COMPS²³) from the Fault Detection and Management (FDM) system. The primary use of ERF is to determine the failures that could cause the observed failure conditions (as evidenced in telemetry data and C&W messages). ERF can also assist a flight controller in evaluating the impacts of a failure, determining if a response to the failure is necessary, and planning needed responses. In particular, a flight controller can use ERF to perform the following activities:

- Predict the consequences of taking no action to correct failure conditions (i.e., failure propagation)
- Assess the impacts of a fault in terms of unavailable hardware and lost functionality, or loss of redundancy in either of these
- Predict the next worse failure (i.e., critical functions affected by fault, such as loss of redundancy introducing a new single point failure)
- Evaluate "what-if" scenarios by permitting the controller to hypothesize a failure or changed sensor value, and analyze its consequences

ERF is a passive fault management system. It does not recommend crew actions in response to failures, and does not issue vehicle commands.

Implicit in these activities are two different modes of operation, real-time data analysis and "what-if" evaluation of consequences. For real-time data analysis, the model is evaluated to identify the causes of failure conditions observed in telemetry data and C&W messages. For what-if evaluation, hypothetical failures or sensor values are postulated to predict the consequences if such failures or sensor values were to occur. These modes correspond to different ways of interacting with the intelligent system. For the real-time data analysis, the analysis of failures is automatic and data-driven. For the what-if evaluation, the analysis of failures occurs at the discretion of the operator and the conditions of the analysis are controlled by the operator. The distinction between these modes of operation, and effective ways of shifting between them, have not yet been addressed in the system design.

Instead of providing typical explanation capability, the collaborative activities described above are supported by having the human and the system "share" a representation. In ERF, knowledge about failure causality is represented explicitly in the failure model digraph. Since the system reasons using the same model of failures that the operator monitors, both the operator and the intelligent system use the same representation of information to perform fault diagnosis. This shared representation assists in understanding failure situations as they develop (e.g., failure events initiate analysis as they occur and the results of this analysis are shown on the digraph) and in establishing expectations about what might happen next (e.g., predict failure propagation paths and next worst failure). Explanation is not needed because the situation is revealed by changes in the representation. Using an explicit representation of the failure model also supports on-the-job training of the operator, by assisting the development of a mental model of system failures. As the ERF representation becomes more widely used in space operations, the effectiveness of this shared representation can be evaluated.

ERF supports the operator in supervising its activities. Having an explicit representation of the failure model used by the intelligent system available to the operator assists the operator in understanding system capabilities and activities. The operator's understanding of what is represented in the knowledge base (i.e., system failures, causal connections) is improved by monitoring the failure model digraph. The operator can also evaluate the effects of system reasoning on the knowledge base by reviewing the nodes remaining in the digraph after failure diagnosis. The ability of the operator to control the intelligent system depends upon the implicit modes of operation discussed above. During real-time data analysis, the diagnosis of failures does not require operator

²³ COMPS are parameter values computed on the ground. These computations are often based on telemetry data.

intervention. During what-if evaluation, the operator controls both the failure conditions and the type of analysis performed by the intelligent system.

This description of how the operator interacts with ERF is focused on its use during real-time operations. ERF is also used to construct and examine failure digraphs in preparation for a mission. It can assist in generating mission-specific procedures as well, by providing a model of system failures for use in evaluating the effects and risks of procedures.

4.4 Supporting User Interface Capabilities

The ERF system is early in the design phase of system development, and the design of a graphical user interface was in progress at the time of the interviews. This user interface design is not yet complete, however, and consists primarily of the FEAT user interface. Although the final design will likely change, the FEAT user interface design is described in this section, particularly how it supports the operator in managing the monitored process and in supervising the intelligent system. Since ERF is a real-time support system, this discussion focuses on the failure model analysis capabilities of FEAT instead of the model construction capabilities.

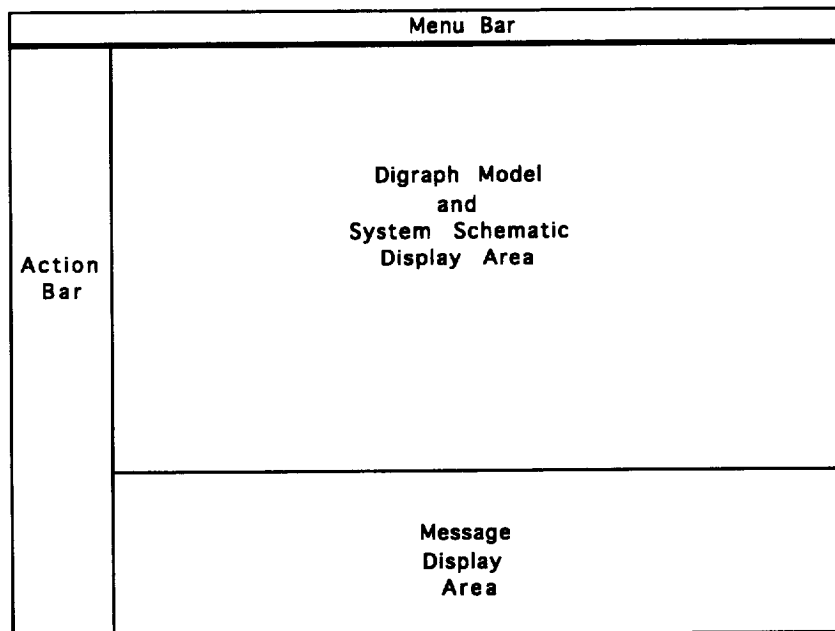


Figure 4-2. The Workspace of FEAT.



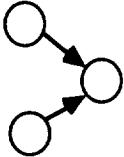
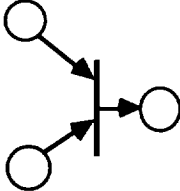

The workspace of FEAT (figure 4-2) consists of the following regions: (1) a graphical presentation of the failure model, (2) an action bar and menu items for operating on the model, (3) a system schematic, and (4) a message list. The model of system failures can be used both to assess failures in the monitored process and to assist the operator in understanding intelligent system activities. Both the action bar and portions of the menu provide capabilities for analyzing this model. The system schematic can be used to associate the failures in the failure model to faulty components in the system. Intelligent system conclusions about failure descriptions and causes are provided in a scrolling message list.

Model of System Failures

The model of system failures is useful both in managing the monitored process and in supervising the intelligent system. To assist in supervising the intelligent system, the operator can monitor and review the failure model to clarify system reasoning and knowledge. To assist in managing the monitored process, the operator can analyze the failure model to determine both the causes and the effects of failures.

Monitoring and analyzing the failure model involves three portions of the workspace: the graphic presentation of the failure model, the action bar, and some items on the menu bar. The graphic of the failure model illustrates information about failure causality in the monitored process. The action bar and menu items provide operator actions for analyzing the failure model. Table 4-1 summarizes the graphic forms used to construct a failure model digraph. An example of the FEAT display illustrating a failure model digraph is shown in figure 4-3.

Table 4-1. Graphic Forms in a Digraph

Symbol	Name and Use
	node; denotes a possible failure point within a model of system failures
	edge; indicates a potential failure path within a model of system failures
	OR relation; indicates that if either node on the left fails, the right-hand node fails
	AND relation; indicates that both nodes on the left must fail (and those failures overlap in time) in order for the right-hand node to fail
	sensor; in the ERF implementation, this symbol represents a sensor
Node Name/Mnemonic Node Description Component Name User Field	text block; text associated with a node, e.g., FDIO1SABE Diode 1 fails with no output DIODE 1 ELECTRICAL

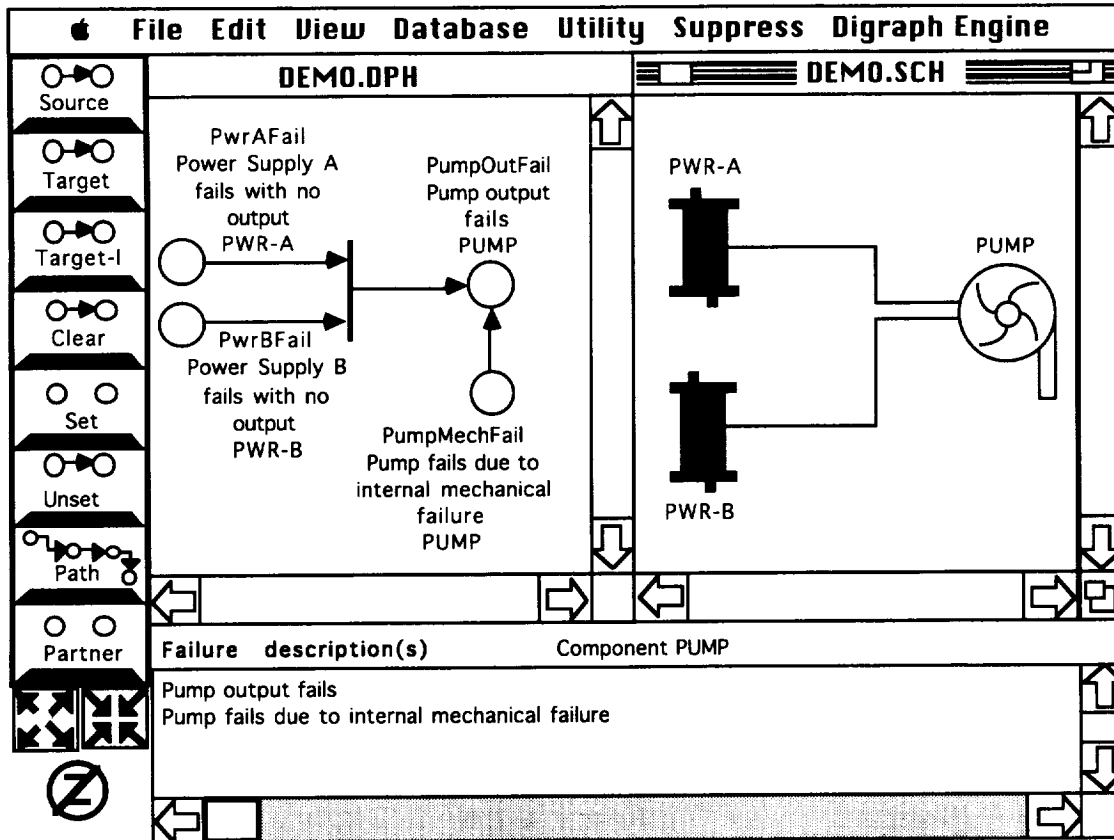


Figure 4-3. Example of the FEAT Display (LESC, 1991).

The user interface shown in figure 4-3 supports both monitoring and controlling the intelligent system. Graphically presenting the failure model supports the operator in supervising the intelligent system by improving the operator's understanding of system capabilities and reasoning. The digraph illustrates the failure causality knowledge in the knowledge base, and shows changes in the state of that knowledge base (e.g., highlighted nodes correspond to candidate failures). The action bar and some menu items provide the operator with intelligent system control capabilities. In addition to the typical types of intelligent system control (e.g., loading a knowledge base [failure model], starting and stopping the system), the operator can change the state of the knowledge base (e.g., setting failure conditions for a hypothetical analysis) and control the type of reasoning performed by the intelligent system (e.g., selecting the type of analysis, such as fault isolation or evaluation of consequences).

A user interface where the operator can graphically view the same failure model that is encoded in the intelligent system knowledge base provides the following advantages:

- Permits the operator and intelligent system to "share" a representation of failure information and, thus, develop a shared view of failure situations (see also section 4.3)
- Assists the operator in monitoring the intelligent system by clarifying its knowledge base and reasoning
- Supports the operator in developing a mental model of system failures while performing fault management tasks

Of course, the final evaluation of the effectiveness of this user interface design will occur during operational testing.

The user interface shown in figure 4-3 also supports the operator in managing the monitored process by providing capability to perform the collaborative, fault management activities described in section 4.3. These activities require two different ways of using the intelligent system: real-time analysis of failure conditions indicated in telemetry data, and what-if evaluation of operator-specified failure conditions. The current user interface design is oriented toward what-if evaluation, since it is based on the FEAT user interface that was designed for off-line analysis. User interface support for real-time data analysis is not well specified at this time.

For real-time data analysis, fault diagnosis is initiated automatically upon receiving data indicating a failure. Failure conditions are propagated through the failure model and the suspected causes of failures are isolated. During the analysis, the sensor nodes corresponding to C&W messages and the failure nodes activated by failure propagation are highlighted on the graphic. When the analysis is complete, only the failure nodes corresponding to the suspected faults are displayed. Although the specifics of operator interaction for data-driven analysis have not been specified, less interaction is needed because data analysis occurs automatically. Operator interaction is likely to focus on evaluating the consequences of the identified faults.

For what-if evaluation, the operator selects the failure conditions to be analyzed and controls the type of analysis performed. The consequences of a hypothetical failure can be evaluated or the possible causes of that failure identified. The operator can initiate the automated analysis capabilities of ERF, or can interact manually with the system to perform what-if evaluation in the following ways:

- Mark a specific node (called the source node) as failed and propagate the failure through the digraph to examine the effects of failing this node (called a downstream analysis, or source propagation). Nodes affected by the failure are highlighted on the digraph graphic and schematic.
- Mark a target node as failed and identify all node failures that could have caused the target node to fail (called an upstream analysis, or target propagation). Nodes that could cause the target failure are highlighted on the digraph graphic and schematic.
- Identify individual failure nodes (singletons) in a failure path that could be solely responsible for the failure of the target. The candidate singletons are highlighted on the digraph graphic and schematic.
- Identify node pairs (doubletons) that must both fail before the target node can fail. The candidate doubletons are highlighted on the digraph graphic and schematic. Note that these failures are not necessarily simultaneous failures (occurring at the same point in time), but must be coincident failures (i.e., have overlapping times of occurrence).
- Identify the other members of the possible doubleton pairs (or partners). The partners are highlighted on the digraph graphic and schematic.
- Identify the region where the paths of two target nodes intersect (called target intersection). The nodes in the path intersection are highlighted on the digraph graphic and schematic.
- Identify a singleton path between two selected nodes. The nodes in this path are highlighted on the digraph graphic and schematic.

To perform these activities, the operator uses capability provided in the action bar to the left of the screen and in the menu bar at the top of the screen. Specific analysis capabilities provided by the action bar are described in figure 4-4. Menu items useful in digraph analysis are described in figure 4-5. Because the real-time support is of primary interest, menu items used for constructing and maintaining a digraph models (e.g., cut, paste, etc.) are not discussed. The color coding for highlighting nodes on the failure digraph is shown in table 4-2.









 Source	Initiate Source Propagation; clicking this button will cause the selected node(s) to propagate as sources through the digraph, and nodes affected by the selected source(s) will be highlighted in red (the source(s) will remain green)
 Target	Initiate Target Propagation; clicking this button will cause any selected node(s) to be propagated as targets and those nodes which are potential targets will be highlighted in red (singletons) or magenta (doubletons)
 Target-I	Initiate Target Intersections Propagation; if more than one target is selected and the failure paths intersect, the singleton intersections will be highlighted in blue and the doubleton intersections in cyan
 Clear	Clear all Selects, singletons & doubletons; removes the highlighting from all nodes except those selected with the Set button
 Set	SET selected nodes and propagate effects; sets the selected node(s) as failed and highlights these nodes and components in yellow
 Unset	Unset ALL SET nodes & selections; removes the setting of all nodes previously selected as "failed" and removes the yellow highlighting
 Path	Find singleton path between selected two nodes; causes the singleton path between selected target and source to be highlighted in red
 Partner	Highlight doubleton partners for TARGET and selected doubleton; after target propagation, distinguishes other doubletons that will pair with a selected doubleton and highlights them in purple

Figure 4-4. Description of Analysis Capabilities Provided by Action Bar.

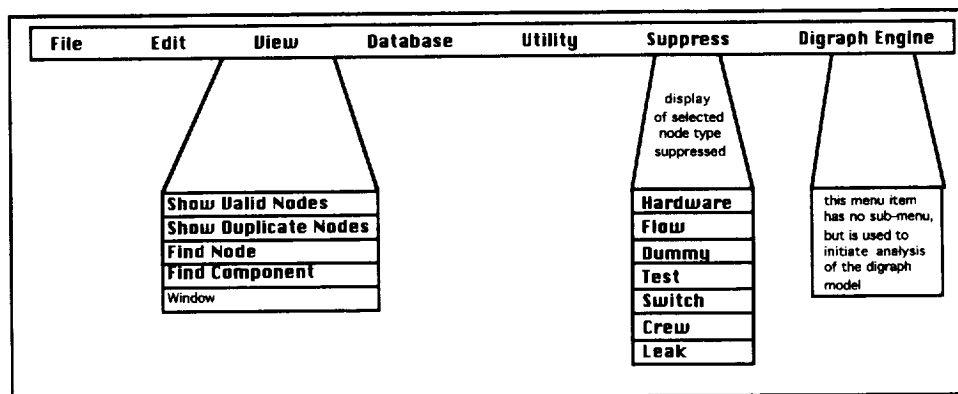


Figure 4-5. Description of Analysis Capabilities Provided by Menus.

Table 4-2. Color Coding Used on the Failure Digraph

Use	Color
Blue	singletons that are common to all target selections
Cyan	doubletons that are common to all target selections
Green	components that are selected
Magenta	doubletons
Purple	associated doubleton partners
Red	singletons to some selected target; singleton to one target and doubleton to another
Yellow	nodes and components affected by the SET function
Black	unhighlighted portions of the digraph model

The FEAT user interface provides an overview window that summarizes the status of each node in the digraph and each component in the schematic. The overview window consists of two lists, one containing each node in the digraph and the other containing each component in the schematic. Icons to the right of each node or component indicate the status of the node with respect to failure analysis (e.g., is the node selected, is the node a partner in a doubleton pair). This status is displayed redundantly, using unique icons in combination with the color codes shown in table 4-2 (see figure 4-6). The overview window can also be used to navigate within a complex failure model or schematic that is larger than the screen viewport. The user can locate a specific node or component by selecting its name in the list using a mouse. See figure 4-7 for an example of the overview window.

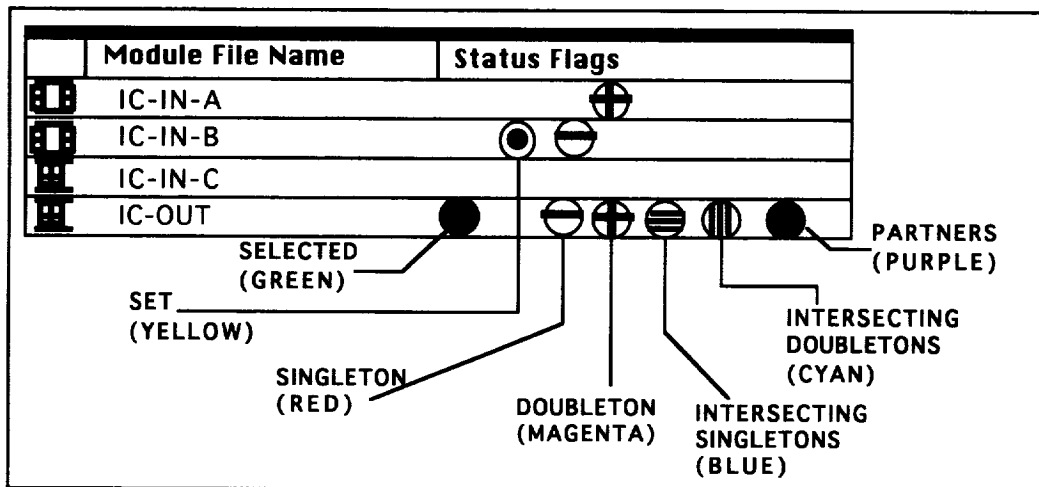


Figure 4-6. Icons and Color Coding used in the Overview Window.

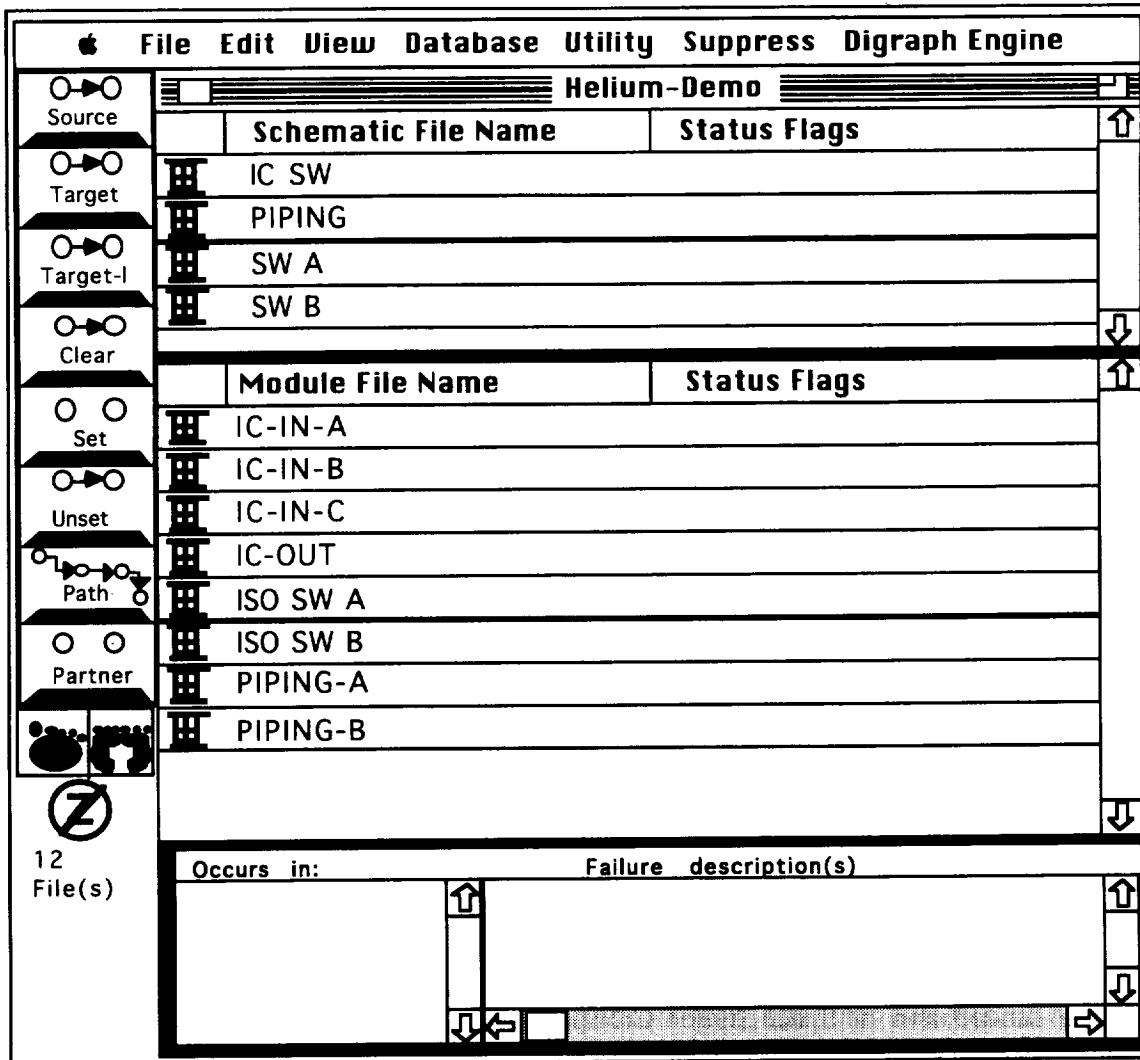


Figure 4-7. An Example of the Overview Window.

As mentioned previously, the ERF user interface design is not complete, and largely consists of the user interface designed for FEAT. Some user interface issues not yet resolved are listed below:

- Integration of information from multiple sources, including telemetry data from the vehicle, C&W messages from the FDM, failure knowledge from the knowledge base, and conclusions about failure states inferred by the system
- Clear distinction between the two modes of operation (real-time data analysis and the what-if evaluation), and methods for shifting between these modes
- Effectiveness of the digraph representation and the schematic representation for operational support

Schematic

A schematic illustrating the physical layout of system components is also provided. The flight controller can select components for analysis using the graphic of the schematic as well as the graphic of the digraph model. This schematic is displayed concurrently with the digraph failure model (see figure 4-3). This arrangement assists the operator in associating the failures shown in the digraph with the corresponding faulty components in the schematic. In fact, the same color coding is used to indicate failures in the digraphs and faulty components in the schematic. This comparison can be useful both in assessing the impacts of the failure to system hardware (and associated functionality), and in evaluating ways to recover from failures (e.g., schematic should indicate whether redundant capability is available).

The schematic and the digraph have similar layouts (figure 4-3). There is not a one-to-one correspondence between failures and faulty components, however. The failure model can also include artificial nodes created as part of the model but having no corresponding physical component.

Message List

The message window lists the possible failures for a selected component. Messages describe failures and identify the possible causes of failures. The schematic component affected by a failure is identified in the header to the message list (e.g., in figure 4-3, messages refer to the PUMP component) and is highlighted in the schematic. Instead of interacting with the digraph graphic, the flight controller can mark failure nodes for analysis by selecting lines of text in the message list.

4.5 Design Process

This project was initiated by the Software Technology Branch/PT4 at JSC to investigate the use of artificial intelligence tools and technologies in space flight operations. This investigation included surveying the technologies in operational use at NASA. Based on the results of this investigation, a project was proposed to modify the off-line analysis tool FEAT for real-time use in the CCC. This project was funded, and the Space Station Ground System Division/DJ (J. Lauritsen) at JSC assumed management of the project. Loral Aerospace, Space Information Systems Division (J. Dell), the CCC prime contractor, became involved at this time as well.

ERF has been designated as baseline software for the CCC. Thus, it must be implemented on a standard CCC hardware platform and must comply with CCC standards and guidelines. The system is also constrained from issuing commands directly, and must instead recommend that a human take any needed action.

ERF is being developed as a Rapid Development Project for the CCC. As such, it must meet the content specification of CCC formal documentation (i.e., must contain all concepts from system analysis and design), but is not required to follow the format specification of the formal documentation. To date, a Requirements Definition Document has been delivered, and both a System Requirements Review (October 1991) and a Preliminary Design Review (March 1992) have been conducted. At the time of the interviews, the developers were beginning user interface design and initial system implementation.

ERF is one of the few systems studied where significant analysis was done before prototyping. The bases of this analysis were descriptions of representative operational situations generated by the users (S. Kirby/DE2). These specific failure situations were generalized as typical failure scenarios. These scenarios were used to identify the functions that must be performed during fault management operations. These functions were represented as Data Flow Diagrams (DFDs) specifying fault management activities and the information required to perform these activities (see

example below). The DFDs were then verified by mentally postulating faults and evaluating whether the resulting activities were reasonable (i.e., conducting a "mental simulation"). These DFDs were used to write the Functional Definition Document, and were included in that document. The DFDs were also an effective means of discussing system capability with the users, and assisted in gaining the understanding and support of the user community.

Example: Figure 4-8 illustrates a DFD of the fault detection portion of the fault management process. The fault patterns provided to the user include single and dual-point failures, and predicted failure chains. The C&W information provided to the user includes predicted C&W cascades and secondary C&W lists. User input includes options and authorizations, parameters unavailable electronically, and the shutdown command. Figure 4-9 summarizes this information exchanged between the flight controller and ERF during fault detection.

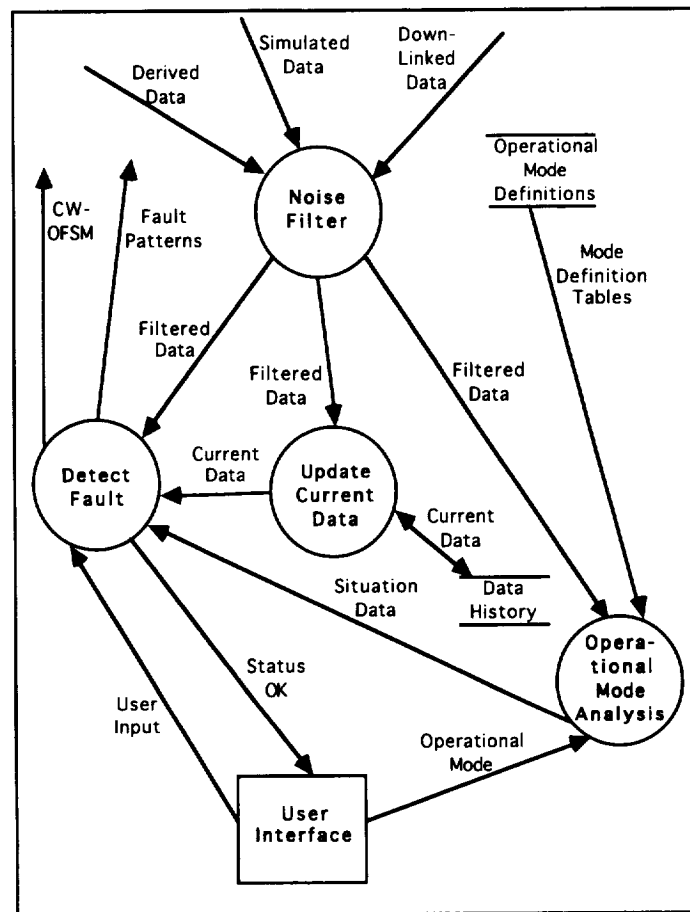


Figure 4-8. Data Flow Diagram of the Fault Detection Process²⁴.

²⁴ Diagram derived from the Fault Detection and Management Functional Definition Document (1991).

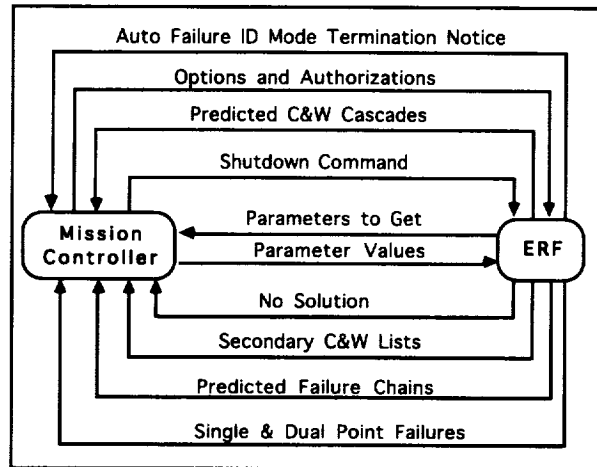


Figure 4-9. Information Exchanged between Flight Controller and ERF.

The scenario-based approach to systems analysis involved the users very early in system development. Users described representative scenarios used in identifying system functionality and information flow needed for fault management. Users also assisted in evaluating these functional specifications using mental simulations. This approach made effective use of user expertise and time by focusing discussions on system use during operations (i.e., "describe how this system will be used") instead of system or technology features (i.e., "list the features the system should have"). The development team also possessed considerable expertise in both space and aviation operations (R. McNenny and C. Clark from MDSSC), as well as artificial intelligence technology expertise (S. Jowers from MDSSC).

4.6 Study Method

Information about ERF was obtained from demonstrations and interviews with S. Jowers and Robert McNenny in November (5) of 1991 and January (17) and July (29, 31) of 1992. The case data sources cited in section 4.7 were also used, and the figures shown throughout section 4 were derived from figures included in these case data sources.

Project Representatives

- Steve Jowers (MDSSC)
- Robert McNenny (MDSSC)

4.7 Case Data Sources

Bohler, Jim. *DA/FA/RM Planning Meeting*. Briefing by Grumman. October 1991.

Clark, C., S. Jowers, R. McNenny, C. Culbert, S. Kirby, and J. Lauritsen. *Fault Management for the Space Station Freedom Control Center. Proceedings of AIAA 30th Aerospace Sciences Meeting & Exhibit*. Reno, NV. January 1992.

DRL LI 25B (MDSSC). *Space Station Control Center Extended Real-Time FEAT Subsystem Functional Requirements*. December 1991.

JSC-13193. *Space Station Control Center User Detailed Functional Requirements*. April 1991.

Lauritsen, J., D. Lawler, and S. Karro. Extended Real-Time Feat in the Space Station Control Center. Briefing at *NASA Monitoring and Diagnosis Workshop*. January 1992.

LESC, 1991. *User's Guide for the Failure Environment Analysis Tool*. Versions 3.0 (Digraph Editor) and 3.3. November 1991.

Malin, J., D. Schreckenghost, D. Woods, S. Potter, L. Johannesen, M. Holloway, and K. Forbus. *Making Intelligent Systems Team Players: Case Studies and Design Issues, Vol. 1. Human-Computer Interaction Design; Vol. 2. Fault Management System Cases*. NASA Technical Memorandum 104738. Johnson Space Center, Houston, TX. September 1991.

Schier, Jim. *Digraph Analysis*. Presentation by Grumman. October 1991.

Schier, J., and D. Gantzer. *Integrated Digraph Analysis of SSF Reboost Function for Stage 2*. Briefing by Grumman. November 1991.

SSP-30000. Digraph Analysis Directive. *Space Station Program Level II Change Request to Section 2, Part 4, Revision B (draft)*. October 1991.

Stevenson, R., and R. McNenny. Fault Isolation in Space Station Freedom Systems from the Space Station Control Center. Presented at *JAIPCC '92 Symposium*. March 1992.

SECTION 5

PDRS DECISION SUPPORT SYSTEM

5.1 System Description

The PDRS DESSY is a fault management system for the Space Shuttle RMS. DESSY assists the RMS/Mechanical System flight controllers in monitoring status and detecting faults for two RMS subsystems, the MPMs and the MRLs. The MPMs rotate the RMS arm away from the vehicle for RMS operations (deploy), and toward the vehicle for storage (stow). The MRLs attach the arm to the body of the vehicle for storage. DESSY monitors telemetry data to assess the state and status of these subsystems.

Because the DESSY case was studied previously (Malin, et al, 1991), it provides an opportunity to evaluate how the system has evolved since the initial study. In this report, changes from the original system are discussed in detail, including both the problems with the original design and how these problems were solved. Monitoring system development over a period of time has also resulted in a better understanding of the intelligent system design process.

The original version of DESSY was developed in CLIPS on a PC. The current version of DESSY is developed in G2 and the XWindows System, and runs on Sun SPARC Unix workstations. It uses telemetry data from the RTDS. It is located both in the RMS Laboratory and in the Multi-Purpose Support Room (MPSR), where it has been used during some missions involving the RMS. DESSY can also run on a MassComp workstation, but has had performance problems on this platform.

5.2 Intelligent System and Functions

The PDRS Decision Support System uses telemetry data (1) to monitor the state of the MPMs and MRLs, (2) to detect faults that affect this subsystem, and (3) to assess the impacts of these faults on functional capability. The PDRS includes four MPMs, one located at the shoulder of the arm and three others located along the arm (fore, mid, and aft attachment points), and three MRLs, co-located with the forward, mid, and aft MPMs. DESSY determines the state²⁵ and status of the MPMs and MRLs, and detects motor failures affecting either system. The ability to assess functional capability has been enhanced since the last case study (see section 5.3). This capability assists the flight controller in determining whether the RMS system can support nominal operations after a fault occurs.

DESSY combines object-oriented data structures with rule-based reasoning. The rules provide logic for monitoring and detecting faults in the MPM/MRL subsystem. The planned DESSY design includes additional software modules for monitoring other hardware subsystems in the RMS.

Within the MPM/MRL software module, the rules are partitioned into sets by system function, including data monitoring, state transition detection, failure diagnosis, expert system control, and user interface (Land, et al, 1992). The system can automatically enable or disable rule sets in response to data problems, or to change system context (e.g., configuration change).

²⁵ The states of the MPMs are deployed, stowed, or in transition between these states. The states of the MRLs are released, latched, or in transition between these states.

Many of the changes to DESSY since the last case study have been in response to data problems encountered during operational use of the system. The DESSY project has both characterized these data problems and designed solutions to them (Land, et al, 1992). The following types of problems have been observed during development and testing of DESSY:

- **Loss of data**

When the Space Shuttle enters the Zone of Exclusion, LOS makes telemetry data unavailable at the ground support consoles. Throughout the LOS period, the RTDS telemetry processor providing data to DESSY cycles through the last four to five seconds of data transmitted. Processing this data (which is often of poor quality) can cause intelligent system errors.

- **Erratic data**

The quality of telemetry data usually degrades near AOS and LOS. Processing this data can cause intelligent system errors.

- **Data lags and irregularities**

Data can lag or exhibit transients due to noise or hardware properties (e.g., switches can "bounce" when flipped, causing transients in data). When watching for a timed event to occur (e.g., deploy), data lag can cause the intelligent system to erroneously conclude that the event failed to occur (or occurred late). Transients can cause the system to conclude an erroneous state or event.

- **Ambiguous data**

Weight constraints onboard the Space Shuttle limit the available sensors. Insufficient sensors can result in ambiguous data, where two states or events cannot be distinguished using the available telemetry (e.g., LOS and guillotining the arm have the same data signature). This can cause the intelligent system to make an erroneous conclusion.

DESSY has taken a unique approach to solving these data problems. Instead of the usual approach of preprocessing the telemetry stream to eliminate bad data, the DESSY rule base was developed to be robust to data problems. Robustness is achieved by designing for graceful degradation when processing bad data and graceful recovery when good data becomes available (Land, et al, 1992). Graceful degradation means minimizing the incorrect conclusions drawn by the intelligent system. This is accomplished by disabling rules when data quality is poor or uncertain, and by considering operational context and expectations when interpreting data. Graceful recovery is accomplished by self-correcting rules (see figure 5-1) that permit retracting erroneous conclusions. Special rules have also been written to handle ambiguities in data that have implications for critical functionality (e.g., dead face rule to distinguish LOS from arm guillotined).

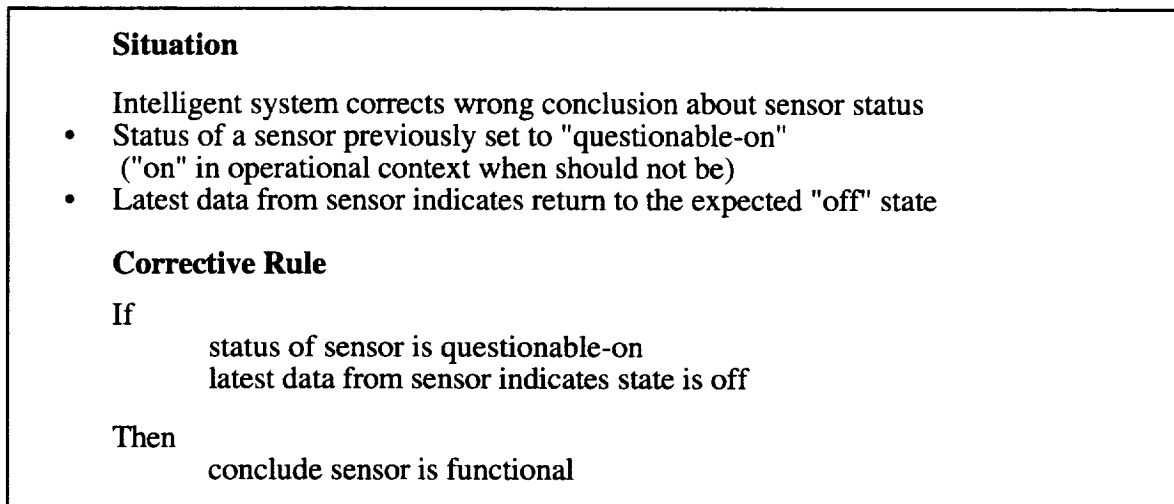


Figure 5-1. Example of a Self-correcting Rule in DESSY (derived from Land, et al, 1992).

5.3 Human-Intelligent System Interaction Functions

The original DESSY HCI design concepts have been updated to provide additional information needed by both the intelligent system and by PDRS flight controllers to perform fault management for the MPM/MRL subsystem. These information items include both telemetry data and intelligent system conclusions. The following information requirements were identified during operational testing of the system:

- **System power**
Flight controllers monitor the state of the payload bay power switch (called mech power) to determine if power is available for RMS operations. Both mech power and the power bus enables are telemetry values from a related, external system (i.e., the electrical power system) that were not identified in the original HCI design concepts.
- **Ops Stats and inferred state of switch on Shuttle**
The Ops Stats are telemetry data values indicating whether an AC power relay is active and the motors are on. Both the intelligent system and flight controllers use the patterns of activation in the Ops Stats to infer that MPM/MRL commands have been issued. An active Ops Stat indicates that MPM/MRL operations might have been commanded, not that a command has occurred²⁶. To infer that a switch commanding MPM/MRL operations has been enabled by the crew, flight controllers look for both the Ops Stats to be on and for motion to be observed by a sensor (i.e., the microswitches).
- **MPM/MRL subsystem status**
The original concept of subsystem status was refined to better support flight controllers in assessing functional capability after a system failure. A composite assessment of subsystem functional capability has been added; the original status only addressed specific components of the subsystem, such as the forward MPM. Additionally, the system now distinguishes between normal operations prior to any failures (i.e., the *nominal* state) and normal opera-

²⁶ These relays are also activated by the crew for non-RMS activities.

tions after recovery from one or more failures (i.e., the *operational* state). This distinction is made to alert the flight controller that a failure has occurred in the past, and that the monitored process has apparently recovered full operational capability (although subtle alterations due to the failure may not be apparent).

In addition to these new types of information, DESSY integrates information from the intelligent system with telemetry data (e.g., the telemetry bubble shown in section 5.4, figure 5-2). A possible future enhancement is the integration of information from other PDRS real-time support systems (i.e., the RMS Position Monitor, the OPS Mode Monitor, and the Temp Mode Monitor; Muratore, et al, 1990) with DESSY.

DESSY supports the user in monitoring its health in two ways. First, the user can check the quality of the data processed by the intelligent system (i.e., check the data quality status provided by the RTDS data acquisition system). Second, the user can check the intelligent systems assessment of its conclusions. As part of handling data problems, DESSY alerts the user when a data value is "questionable" and intelligent system conclusions might be adversely affected.

Although the original design concepts included considerable capability for the user to intervene in intelligent system processing²⁷, DESSY provides limited support for such intervention. Instead, DESSY has been designed to be robust with minimal human intervention. For example, the partitioning of the rule base enables the intelligent system to automatically disable portions of its knowledge base to prevent reasoning about bad data. Although only the intelligent system can disable a rule base at this time, this architecture would support adding capability for a flight controller to manually disable rule sets.

In cases where the intelligent system has failed irrevocably, the human can take over all intelligent system tasks. The system design supports such take over by making access to telemetry data independent of the intelligent system, and by putting information from the failed intelligent system visually in the background (i.e., greying out portions of the display showing information from the intelligent system). See section 5.4 for an example of the display design supporting this capability.

Although DESSY does not support real-time playback of telemetry as described in the original design concepts, users can replay previously recorded data for use in off-line analysis. This has been very useful in testing the system.

5.4 Supporting User Interface Capability

The original HCI design concepts for DESSY included a hierarchy of screens, corresponding to the PDRS hardware subsystems. A system summary screen, or Integrated Status Display, was at the top level of this hierarchy, and the MPM/MRL Subsystem Status display was in the second level. The user viewed one screen at a time. Figure 5-2 shows the original HCI design concepts for the MPM/MRL Subsystem Status display. As shown in figure 5-3, the workspace design of DESSY has been modified such that portions of the Integrated Status Display containing key summary information and the user control buttons are shown in conjunction with the MPM/MRL Subsystem Status Display. This workspace consists of (1) a mission header at the top, (2) control buttons just beneath the mission header, (3) a status summary panel for PDRS subsystems and related external systems²⁸ at the left of the screen, (4) a region to the right that displays detailed

²⁷ User intervention capability in the original design concepts (Schreckenghost, 1990) included (1) restarting from checkpoint, (2) correcting erroneous conclusions and parameters internal to the intelligent system, (3) providing the system with new information, (4) enabling and disabling rule sets, and (5) directing the system to focus on a rule set.

²⁸ The information requirements for illustrating overall system status are not complete.

status information from the MPM/MRL subsystem, and (5) a message list at the bottom. This approach will assist flight controllers in monitoring the status of several PDRS subsystems while looking at the details of a specific subsystem, making DESSY more useful throughout RMS operations.

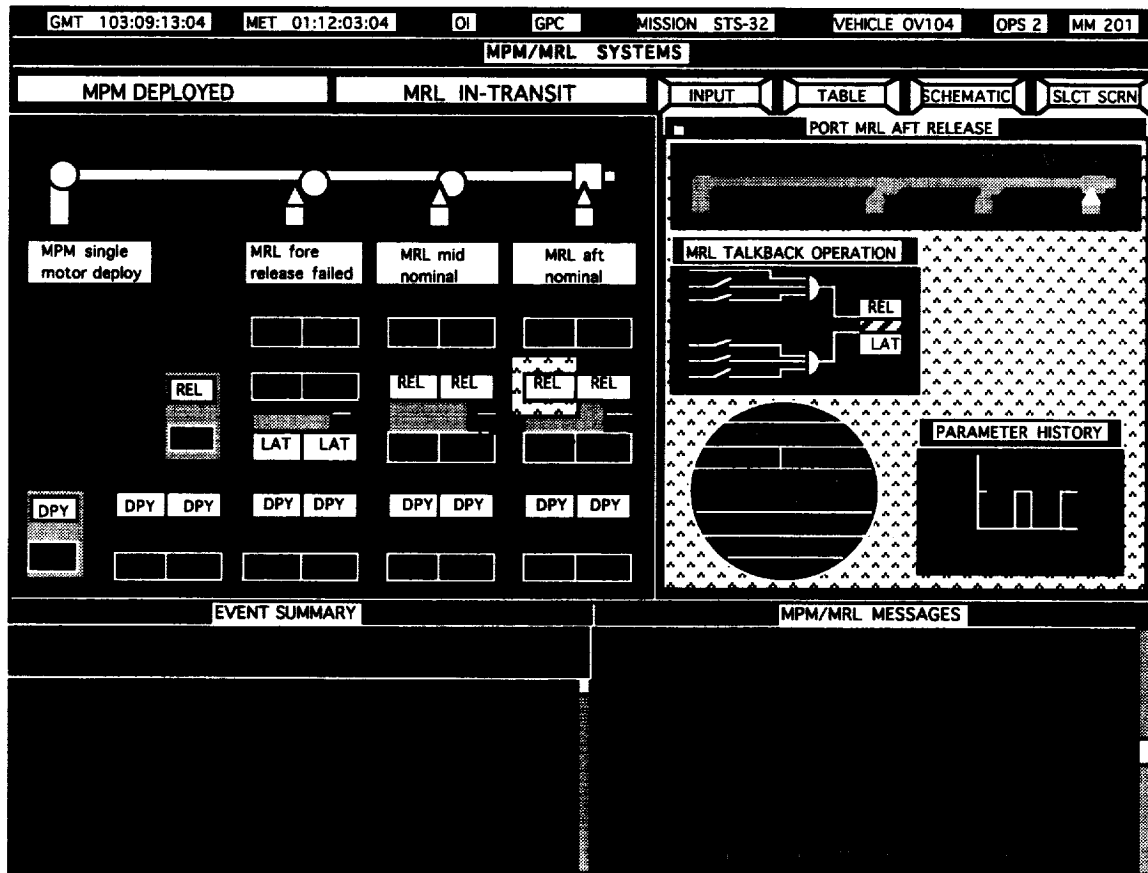


Figure 5-2. Original HCI Design Concepts for the MPM/MRL Subsystem Status Display

DESSY supports the operator in monitoring its activities. Information assisting the operator in monitoring the health of the intelligent system is displayed in two areas. The quality status in the mission header displays the assessment of data quality over the last 100 cycles by the RTDS data acquisition system. "Questionable" intelligent system conclusions (i.e., conclusion possibly erroneous) are annotated as such in both the message list and the subsystem status panels at the top of the dynamic region.

As described in section 5.3, DESSY is designed for robustness with minimal human intervention. If the human does have to intervene, however, the DESSY user interface supports that intervention. When the human takes over intelligent system tasks, the user interface can reflect this task reallocation by showing regions of the display with information from the intelligent system in grey (see figure 5-4). This design focuses operator attention on the displays showing data (telemetry and COMPS) needed to do intelligent system tasks manually, and puts information from the intelligent system into the visual background. The operator uses the "sys cntrl" button to grey out information produced by the intelligent system.

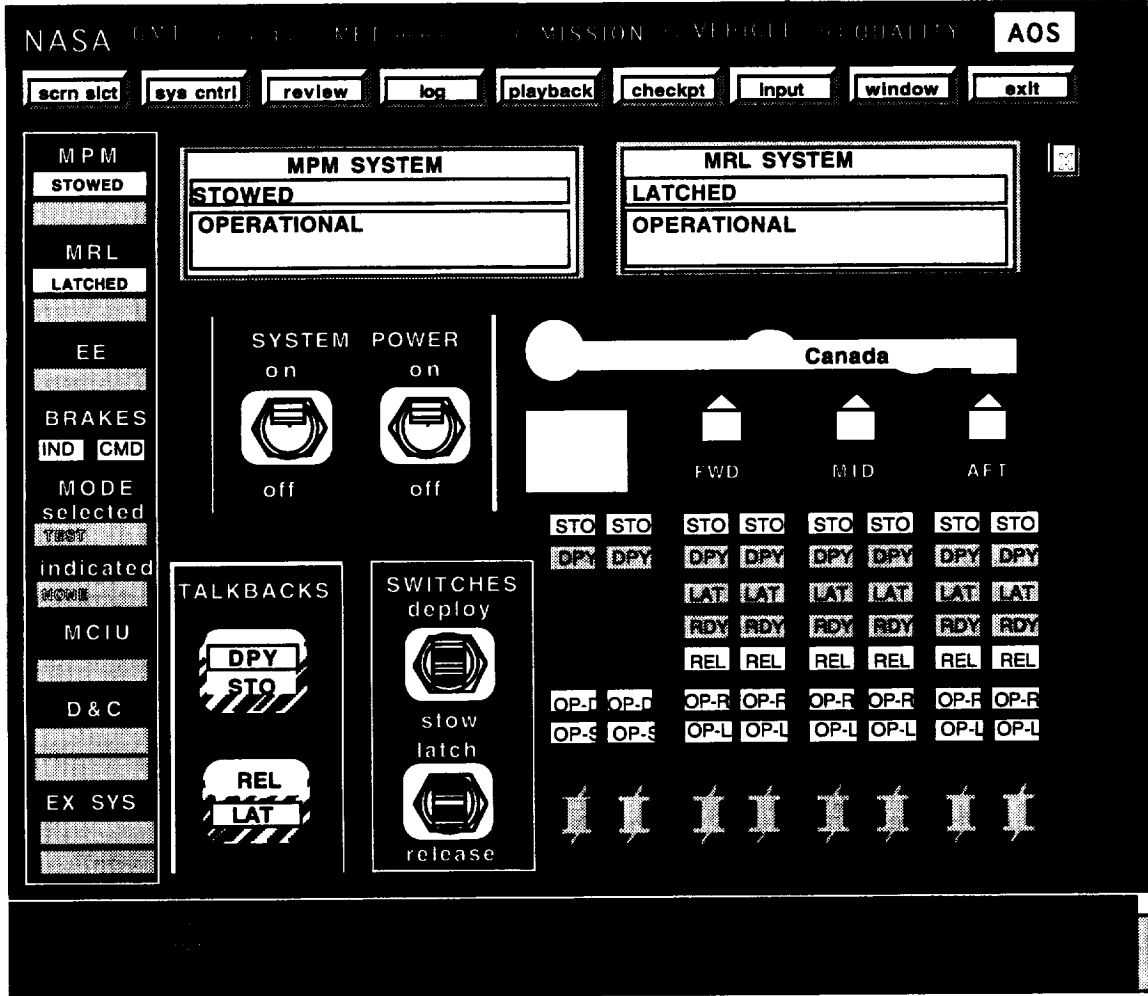


Figure 5-3. Information from Integrated Status Display Combined with MPM/MRL Subsystem Status Display.

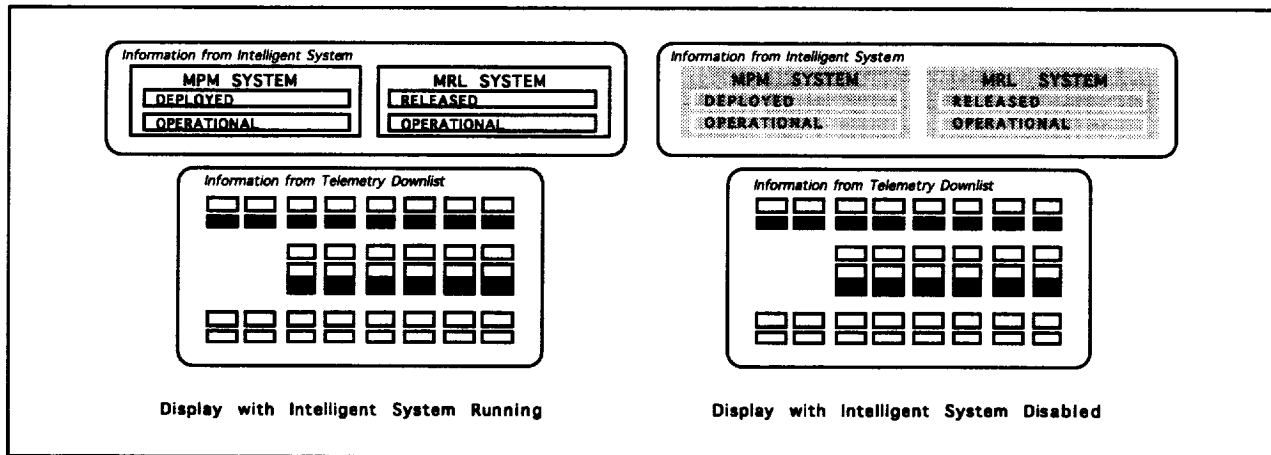


Figure 5-4. Illustration of a Portion of DESSY User Interface After Manual Take Over.

Many new information items have been added to the MPM/MRL display since the original design concepts (see section 5.3). At the request of the flight controllers using DESSY, the display of information about the state of some physical items (e.g., the state of switches and talkbacks onboard the vehicle) emulates the appearance of the physical item. The following describes how these items are shown on the screen:

- System power**
The state of the mech power switch is shown to the left of the dynamic region. A graphic form mimicking the physical switch onboard the vehicle is used to illustrate switch state. Power bus status is shown using color-coded icons positioned beneath the corresponding MPM/MRL components.
- Ops Stats and inferred switch state**
Color-coded panel lights showing the states of the Ops Stats (on or off) are positioned beneath the corresponding MPM/MRL components. The inferred switch state is shown using the switch icons near the bottom of the dynamic region. The icons illustrating switches and talkbacks emulate the Display and Control Panel onboard the vehicle that the crew uses to monitor status and enter commands.
- MPM/MRL subsystem status**
In the original design concepts, the overall subsystem status was shown using color coding on the panels displaying MPM and MRL states²⁹. In the updated design, these panels showing subsystem state have been enlarged to include text messages describing the functional status of the subsystem as well. Subsystem status is redundantly coded using color.

Other user interface upgrades include (1) adding an indicator of AOS and LOS in the mission header; (2) changing the order of the MRL microswitch light panels from the original design to correspond to the arrangement of the talkbacks onboard the vehicle, since the flight controllers are familiar with this ordering; (3) simplifying the user actions for making user log entries (using the "input" button); (4) segregating the development features of the user interface from operational features of the interface; (5) accessing raw telemetry data via pop-up window instead of providing

²⁹ The nature of the status was described in the status panels located beneath each component of the MPM/MRL subsystem.

a separate screen for telemetry display; and (6) using additional colors to code information about the state of arm and user confidence in intelligent system reasoning. Refer to figure 5-2 for the original MPM/MRL Subsystem Status display, and figure 5-3 for the modified display. See table 5-1 for the modified color coding conventions used in DESSY.

In implementing the original design concepts, DESSY has reused some user interface designs and code developed for other RTDS intelligent systems. Reuse of designs and code reduces development costs, avoiding duplicative effort and taking advantage of the experience gained in designing other systems. Additionally, design consistency (e.g., look and feel) across flight support systems is a natural by-product of this approach. To provide the message list capability shown in figure 5-2, available code written using the XWindows System for a message list was integrated with the DESSY G2 user interface. Some user interface designs in DESSY have potential for reuse as well. For example, a pop-up timer window provides the ability to monitor elapsed time (see figure 5-5). The design concept of "greying out" display regions associated with a disabled intelligent system also has potential for reuse (figure 5-4).

Table 5-1. Color Coding in the PDRS DESSY Displays

Color	Use
green	nominal status
yellow	degraded status
red	failed status
dark blue	active item
light blue	inactive item
orange border	Loss of Signal (LOS)
white	arm latched or ready for latch
grey	<ul style="list-style-type: none"> arm uncradled (on arm icon) suppress display of intelligent system conclusions because they are suspect by user (on status panel)

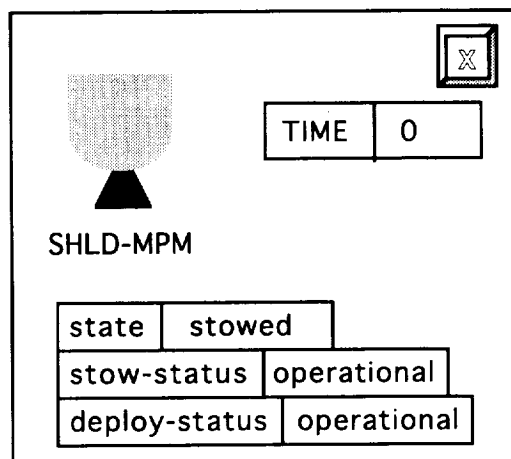


Figure 5-5. Example of the DESSY Timer Pop-Up Window.

5.5 Design Process

DESSY is a RTDS prototype being developed in a joint project by the Intelligent Systems Branch of the Engineering Directorate (J. Malin/ER2 and S. Land/ER2) and the RMS Section of the Mission Operations Directorate (D. Pallesen/DF44). Like most RTDS prototypes, DESSY is being developed iteratively and evaluated in an operational environment. The users actively participate in design (D. Culp, formerly RSOC), both in specifying system requirements and in evaluating how effectively the prototype supports flight controller tasks. The software is written by an artificial intelligence technology expert (S. Land/ER2), and HCI design concepts are developed and evaluated by experts in this area (C. Thronesbery/MITRE, D. Schreckenghost/MITRE, and G. Johns/formerly MITRE).

An important lesson learned during the development of the DESSY system is the importance of scoping the system to be *continuously useful*. To be continuously useful, a system should support the operator throughout operations, since frequent use improves user acceptance of and reliance on the system. Such scoping can include providing information about nominal events as well as failure events to assist the user in understanding situations as they develop. The system must also solve a problem that is well-recognized by all members in the user community, not just a few that are actively involved in the project.

Because the authors have participated in the design of DESSY, this project has provided a unique opportunity to learn about the intelligent system design process. Many concepts for designing an intelligent system prototype for operational use (called "operational prototyping"; Thronesbery and Malin, 1992) have been developed and tested during the design of DESSY. Central to the operational prototyping concept is the need to define task-based information requirements. Much of the effort expended early in prototyping is to discover these requirements. The initial information requirements for DESSY were summarized in a storyboard of HCI design concepts (Schreckenghost, 1990). Further requirements were identified and refined during operational testing of the system, emphasizing the importance of such testing. An important distinction in the types of testing was also made while working with DESSY. Typical user interface testing focuses on the *usability* of the design (i.e., testing that the user interface presents information effectively). The testing of DESSY has also included *utility* testing (i.e., testing that the system provides the right information to support user tasks). Distinguishing between what information is needed and how that information should be presented was also useful in resolving design problems. When these problems occur, separate discussion of the information issues and the presentation issues results in more effective use of the design team.

5.6 Study Method

Information about DESSY was obtained from demonstrations and interviews conducted in January (22, 28), February (5), April (1), May (20, 27), June (30), July (2), and August (7, 14) of 1992. These interviews were conducted with the project representatives listed below. The case data sources cited in section 5.7 were also used, and the figures shown throughout section 5 were derived from figures included in these case data sources.

Project Representatives

- Sherry Land (Intelligent Systems Branch, NASA/JSC)
- Don Culp (formerly RSOC)
- Carroll Thronesbery (MITRE)

5.7 Case Data Sources

Culp, Don. Interpretation of Space Shuttle Telemetry. *Proceedings of the First CLIPS Conference*. August 1990.

JSC (NAS9-18000, TD383). *Payload Deployment and Retrieval System Overview Workbook*. PDRS OV 2102. February 1988.

JSC (NAS9-18000, TD385). *PDRS Malfunction Workbook*. PDRS MAL 2102. April 1988.

JSC (NAS9-18000, TD386). *Payload Deployment and Retrieval System Nominal Operations*. PDRS NOM OPS 2102. December 1988.

Land, Sherry. *Phase I Technical Intern Review*. Automation and Robotics Division, Engineering Directorate, Johnson Space Center. August 1991.

Land, S., J. Malin, and D. Culp. DESSY: Making a Real-time Expert System Robust and Useful. *Proc. NASA Space Operations, Applications, and Research Symposium*. Houston, TX. August 1992.

Malin, J., D. Schreckenghost, D. Woods, S. Potter, L. Johannesen, M. Holloway, and K. Forbus. *Making intelligent systems team players: Case Studies and Design Issues*. NASA Technical Report 104738. Johnson Space Center. Houston, TX. September 1991.

Muratore, J., T. Heindel, T. Murphy, T., A. Rasmussen, and R. McFarland. Real-time Data Acquisition at Mission Control. *Communications of ACM*. December 1990.

Schreckenghost, D. *PDRS Intelligent System Human-Computer Interface Design Concepts*. Briefing at Johnson Space Center. July 1990.

Thronesbery, Carroll. *Analysis and Design of Intelligent Systems: The Role of Operational Prototyping*. Presentation at JSC Training Class. June 1992.

SECTION 6

FUEL CELL MONITORING SYSTEM

6.1 System Description

The FCMS is a fault management system for the Space Shuttle fuel cells and associated power busses (both alternating current [AC] and direct current [DC]). The FCMS assists the EECOM flight controller in detecting, diagnosing, and recovering from faults affecting these fuel cells. It monitors telemetry data to assess the operational status of these electrical components, and recommends crew and flight controller actions based on this assessment. The FCMS can detect multiple, independent faults simultaneously.

The FCMS was developed in G2 and runs on a DEC station using Unix and XWindows. There is also a version that runs on a MassComp Workstation. The FCMS is in use in a MPSR of the Mission Control Center (MCC).

6.2 Intelligent System and Functions

The FCMS uses telemetry data to monitor the state and status of the three onboard fuel cells and the associated AC and DC busses, and to manage faults in these subsystems. The intelligent system is built around the following set of fault categories that were identified by the EECOM flight controllers:

- Main DC Bus
 - High or low voltage
- AC Bus
 - Phase shift (three-phase motor stopped)
- Fuel Cell
 - Coolant pump failures (low delta P)
 - Reactant valve closure
 - H₂O high pH
 - Stack high or low temperature
 - Amps high or low (NOT IMPLEMENTED)
 - Volts high or low (NOT IMPLEMENTED)

The FCMS architecture partitions the knowledge base into segments based on these fault categories. For each fault category there is a corresponding set of fault management rules, called an algorithm. Each algorithm is implemented as a separate process, and can run independently and concurrently with other algorithms. Within a fault category, algorithms can address multiple, independent faults (e.g., detecting both value high and value low), but do not address fault propagation and multiple, dependent faults.

Currently there are a total of 18 algorithms in the FCMS: three for each of the six fault categories that have been implemented. These 18 algorithms contain approximately 50 rules that assist the flight controller in fault diagnosis and recovery. There are an additional 30 or so rules to assist in display management, message maintenance, and other system overhead requirements. These 80 rules are used by the system to deal with known faults and well-defined recovery procedures.

An algorithm initiates fault diagnosis when specific fault conditions are detected (e.g., value out of limits). Steps are recommended that the crew or flight controllers can take to diagnose and recover from the fault. These steps usually include verifying sensor readings and (if possible) reconfiguring the system to compensate for any lost functionality. The FCMS assists flight controllers in performing the following fault management tasks:

- **Fault Detection**

During normal operations, active algorithms monitor incoming telemetry for data values that are out of limits, indicating a suspected fault.

- **Fault Isolation**

When the system receives data that indicate a suspected fault, the monitoring algorithm suggests actions for the crew to take to isolate the fault. As those actions are taken, the system continues to monitor telemetry data to verify that data values change as expected, indicating that the suspected fault is causing the problem.

- **Fault Recovery**

When a suspected fault has been verified, the algorithm suggests specific actions for the crew or flight controllers to take to either recover from the fault or to shut down the affected component. These actions are specific to fault recovery and are not the same as the suggestions for fault isolation mentioned previously. The algorithm then monitors telemetry data for an indication that the suggested recovery actions have been completed, indicating that recovery is complete. An algorithm expects to receive an indication that fault recovery actions have been completed within a pre-specified time. If such an indication is not seen, a second message is issued reminding the flight controller of the suggested actions. Subsequently, if there is still no indication of the actions being completed, the system turns off the monitoring algorithm.

Like most systems used for real-time flight support, the FCMS has problems with the availability and quality of telemetry data. Data is periodically unavailable due to LOS. Failure of a multiplexer/demultiplexer (MDM) also results in a partial data set, due to loss of data from sensors connected to the failed MDM³⁰. Data quality problems resulting in false alarms have also been a problem. Data quality degrades near LOS. Although rarely occurring, a fuel cell shutdown results in poor quality data. To minimize false alarms, the FCMS detects situations where data quality is likely to be degraded. When a fuel cell shuts down, the FCMS automatically disables all algorithms for that fuel cell to avoid errors due to bad data. The FCMS also uses the data quality indicator from the RTDS data acquisition system to detect data problems. When data quality is not perfect³¹, the FCMS assumes LOS and automatically aborts active algorithms (although problems can still occur, since some bad data have already been processed). All algorithms are aborted, even if partway through a fault management sequence. Algorithms are not restarted again until AOS (AOS occurs when data quality is perfect). Because a restart resets the knowledge base, all conclusions made before the restart are lost. Thus, a restarted algorithm may either re-discover a fault detected prior to LOS (and issue duplicate messages and recommendations), or may miss it if no evidence of the fault is seen in data received after the restart (e.g., crew fixed problem during LOS). In addition to these automatic responses, the operator has the ability to manually abort and restart specific algorithms if data problems are observed.

³⁰ In an earlier version of the FCMS, the system detected MDM failures and shut down the associated algorithms. Recent changes in the RTDS data acquisition software have removed access to the data required to detect MDM failures.

³¹ Data quality is not perfect when the value of the quality indicator is less than 100, indicating that at least one data frame in the last 100 cycles was bad.

6.3 Human-Intelligent System Interaction Functions

The FCMS is designed to assist flight controllers in managing faults in the Space Shuttle fuel cells. Fault management using the FCMS is a collaborative effort between the operator and the system. Like many intelligent systems in the case study, the FCMS assesses the state of the monitored process and alerts the operator of process anomalies indicated by out-of-limit conditions. State information is used to annotate system schematics and alarm messages are listed chronologically. Plots of related telemetry data can be accessed to illustrate the evidence for an intelligent system conclusion. The FCMS, however, provides capability beyond most intelligent systems studied by also assisting the operator in fault isolation and recovery. Actions for isolating suspected faults are recommended via a separate diagnostic message list. Actions for recovering from faults are recommended via a third message list (the operator display). As these recommendations are enacted, the system monitors telemetry data for evidence that these actions have been executed successfully.

Although the FCMS can recommend that actions be executed, it does not actually issue commands (i.e., the flight controller must take action, or request that the crew take action). Thus, the flight controller has the option of ignoring system recommendations. However, if the system gets no indication of the successful completion of the recovery actions that it suggests, it will stop the associated algorithm. When an algorithm has been automatically stopped in this manner, the flight controller must manually restart it. If a previously discovered fault situation has not been resolved, this restarted algorithm will rediscover the same fault and issue duplicate suggestions for recovery.

The FCMS supports the operator in supervising its activities. As described previously, the knowledge base is partitioned into separate algorithms, and the operator can monitor and control each algorithm independently. The operator can monitor the status of an algorithm, where status denotes if the algorithm is running or is turned off. Additionally, if the algorithm is turned off, the operator can determine why (e.g., automatically shutdown at LOS). The operator can also control algorithms singly, or can control the complete set of algorithms for each fuel cell. The ability to selectively disable and restart portions of the knowledge base is one of the more interesting aspects of this case, because intelligent systems typically being built by NASA do not provide for such selective control. Selectively disabling portions of the knowledge base provides for graded control of the intelligent system. Part of the system can be restarted or disabled while the remainder of the system operates as usual. The ability to control a portion of the knowledge base can be used by the operator to respond to intelligent system error situations as follows:

- Minimize the adverse effects of bad or unavailable data on the intelligent system
- Respond to unexpected situations that cause intelligent system errors
- Continue operating when part of the knowledge base is erroneous

The FCMS knowledge base is partitioned such that there are no dependencies between algorithms, i.e., no information is exchanged between algorithms. This permits gracefully shutting down an algorithm by minimizing the need to "clean up" after disabling an algorithm. Any interim or erroneous conclusions left over from the suspended processing do not affect the other portions of the system. Once the algorithm is restarted, the knowledge base is cleaned up by returning to a default configuration.

6.4 Supporting User Interface Capabilities

The workspace of the FCMS user interface (figure 6-1) consists of three areas: (1) a dynamic region in which the user selects what is displayed, (2) a set of three message lists, and (3) system configuration buttons. The dynamic region can display the algorithm control panel, a subsystem schematic, or a high density data display. The algorithm control panel is used to monitor and control the fault management algorithms. The subsystem schematics are used to monitor the state and

status of the components of the fuel cells and the main AC and DC busses. The high density data displays show relevant telemetry, similar to the tabular data display currently used for mission support. The message lists categorize intelligent system messages to support different fault management tasks. System configuration buttons are used to configure both the intelligent system and the user interface workspace. Each of these is discussed below.

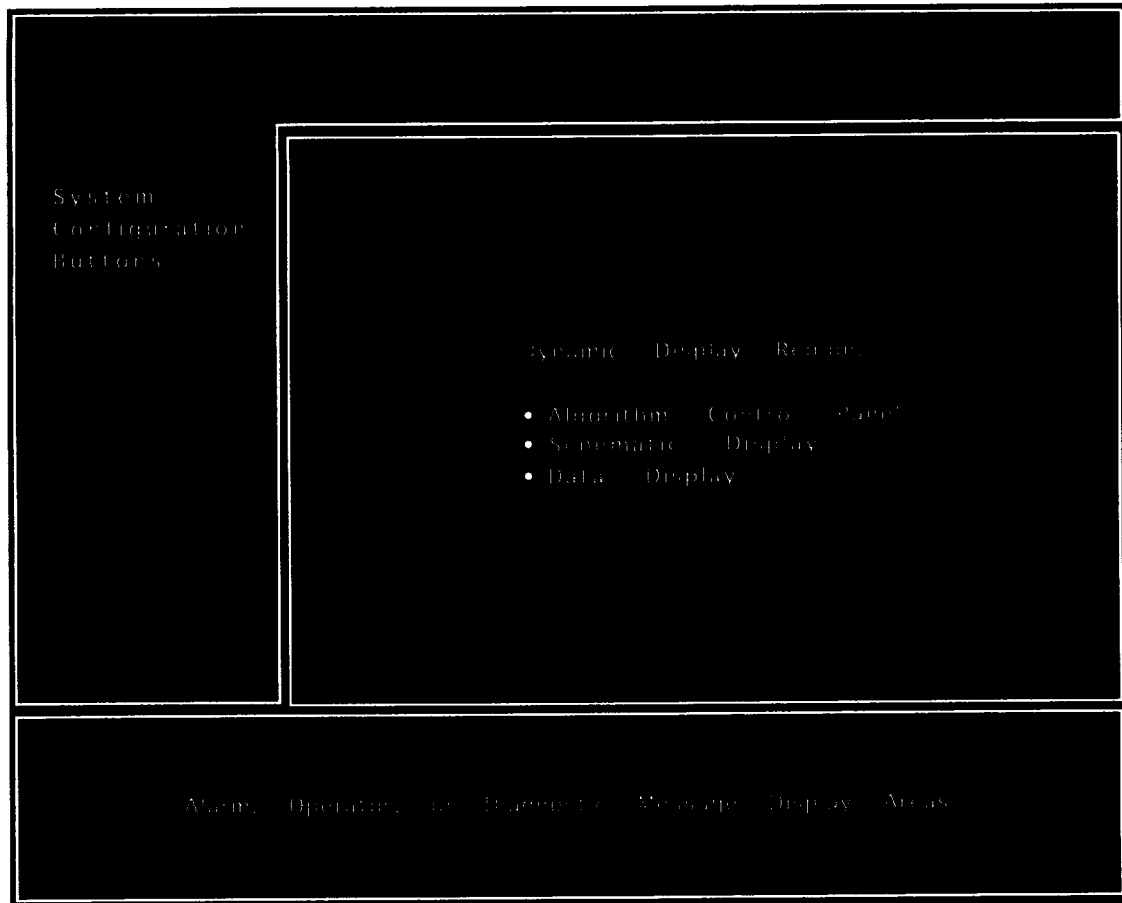


Figure 6-1. FCMS Workspace Layout.

Algorithm Control Panel

As mentioned in the previous section, the FCMS supports user supervision of the system. The algorithm control panel (figure 6-2) supports the flight controller in monitoring the status of each algorithm and controlling its activity. Within the Algorithm Control Panel, algorithms are grouped under the eight fault categories listed in section 6.2. Each category has three algorithms associated with it (one for each fuel cell or bus). For each algorithm, there is a status descriptor and a check box used to control the algorithm. The status descriptor indicates whether the algorithm is active or disabled, and gives some indication of why it was disabled. When the algorithm is active and monitoring data normally, the status descriptor is set to "looking." When the algorithm is disabled, the status descriptor indicates one of the following:

- Stopped - the algorithm was manually disabled
- LOS - the algorithm was automatically disabled at LOS

- FC shutdown - the algorithm was automatically disabled at fuel cell shutdown

Note that these status values also support managing the monitored process by identifying when a fuel cell has been shut down, and when LOS has occurred.

Controlling algorithms is a two-step process. First, the operator chooses the check box near the selected algorithm. Next, the operator chooses either the "Start" button to enable the algorithm, or the "Stop" button to disable the algorithm (these buttons are located near the top of the display). The operator may "Cancel" a previous action as well. This panel also provides the user with window management capability ("Hide"; "Console" to return to a start-up display) and testing capability ("Test" to use simulated data to test algorithm).

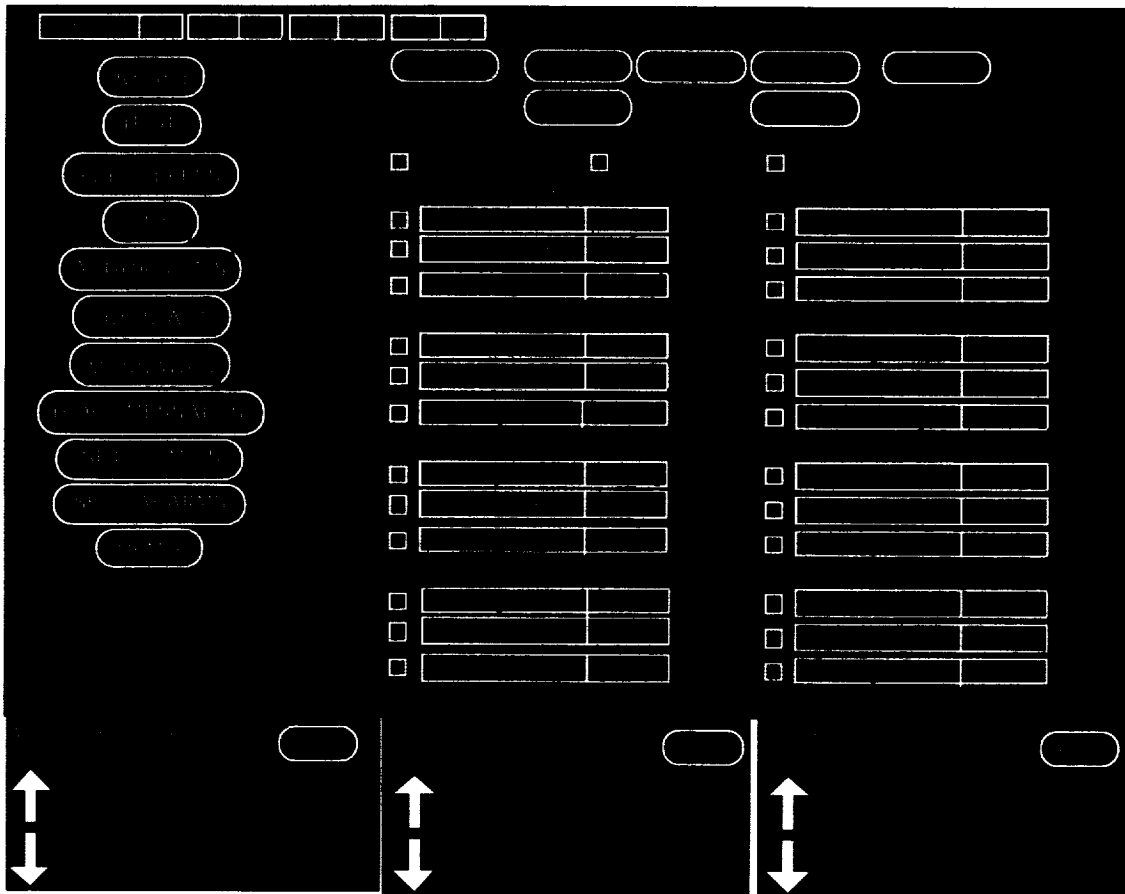


Figure 6-2. The Algorithm Control Panel.

Schematics

Subsystem schematics are provided for each fuel cell, and for the main AC and DC busses. Schematics are annotated with telemetry data. Data values are displayed using both a telemetry bubble display format and a tabular display format. These forms are placed on the schematic near the source of the data. The tabular displays resemble the high density displays currently used for mission support, except that data are organized around the physical components of the system. As seen in many other systems studied, plots of data are also accessible from the schematic via pop-up

window. These plots show the last two minutes of data. As originally designed, the system was also able to annotate schematics with information about MDM failures (a red flag was placed on each telemetry bubble affected by the failed MDM). When the RTDS moved to a new software version, the data necessary to recognize an MDM failure was no longer available to the FCMS.

Message Lists

During fault management, each algorithm issues three different types of messages: alarms, diagnostics, and recommendations. When multiple algorithms are active, a large number of messages are generated. To help flight controllers manage these messages, messages are categorized by the type of task they support: alarms for fault detection, diagnostics for fault isolation, and operator messages for fault recovery. The following message lists are provided, one for each type of message:

- Alarm messages - announce the violation of a data limit
- Diagnostic messages - provide information supporting fault diagnosis
- Operator messages - recommend crew or flight controller actions

Operator messages also have a criticality associated with them. "Normal" operator messages contain recommended actions. "Urgent" operator messages warn that earlier recommendations have not been followed.

Color is used in a variety of ways in the message lists. Messages are color-coded by category of message. The criticality of operator messages is indicated using color as well. Color is also used to indicate that a message has been acknowledged. Color-coding conventions for messages are as follows:

- Black text on yellow background - alarm message, urgent operator messages
- Yellow text on blue background - normal operator message
- Red text on white background - diagnostic message
- black text on green background - acknowledged message

Message lists can be manually displayed or hidden using the system configuration buttons at the left of the screen. Message lists are automatically displayed when a message is posted.

There are difficulties in using message lists for displaying information from concurrently executing algorithms (processes). Since all messages of a given category are posted to one message list, messages from different algorithms are interleaved chronologically. In situations where multiple faults are diagnosed simultaneously, it is difficult to determine which messages are associated with which fault. It is also difficult to see causal relationships between messages, since temporal proximity does not imply relatedness and related messages can be physically separated in the list. This system has a further problem with messages that are posted out of order if the operator is scrolling at the time the message is generated.

System Configuration

The system configuration region on the display (figure 6-1) is used both to configure the intelligent system and to configure the user interface. Intelligent system configuration capability includes the following:

- Reset - resets the knowledge base
- Set Limits - sets the data limits used to trigger algorithms
- Set Alarms - sets the data limits used for annunciating alarms

Configuring the user interface involves managing the windows displayed on the screen. Workspace management capability includes the following:

- Hide - hides the window currently displayed in the dynamic region
- Fuel Cells - displays the fuel cell schematic window in the dynamic region
- EPDC - displays the Electrical Power Distribution and Control schematic window in the dynamic region
- Algorithms - displays the Algorithm Control Panel window in the dynamic region
- Displays - displays the high density tabular displays of data (mimics current Space Shuttle telemetry displays)
- Messages - displays the three message lists
- Hide Messages - hides all message lists

The Demo button is an artifact of the development process and is used to activate a canned demonstration of the system.

6.5 Design Process

The FCMS is in many ways a typical RTDS prototype development project, using iterative prototype development and evaluation in the operational environment. The flight controllers did the initial work in specifying the algorithms necessary to develop the sequences of actions that are used in the FCMS and were involved at various points in reviewing the layout of the displays. However, because of flight controller turnover, there has been less user involvement in later FCMS development activities. This lack of user involvement has also had some impact on system acceptance and the feeling of ownership on the part of the flight controllers. At the date of the last interviews (August 1992), the system was being evaluated in the MCC MPSR during missions and simulations. More use in the MPSR will provide feedback necessary to make the system more robust and more useful.

This system began as a prototype in the Workstation Prototyping Lab in the Mission Support Directorate, and later became an RTDS system in the Mission Operations Directorate. Charlie Robertson of McDonnell Douglas and Curtis Welborn of NASA Mission Support Directorate (the first task monitor) were the initial system co-developers. When Welborn left NASA, Janet Lauritson took over the task-monitoring responsibilities. The system later transferred to RTDS and Troy Heindel became the task monitor. Larry Minter, in the Electrical Systems Section of the Electrical and Environmental Systems Branch was the flight controller who provided input on the algorithmic data, advice on initial display layouts, and expert consultation on operational issues. Rami Al-Ayoobi was not involved in the initial development, but is the current flight controller point of contact and user representative.

6.6 Study Method

Information about the FCMS was obtained from demonstrations and interviews with C. Robertson (MDSSC) and Jerry Snider (MDSSC) in March (9, 18) and August (14, 18) of 1992. The case data sources cited in section 6.7 were also used, and the figures shown throughout section 6 were derived from figures included in these case data sources.

Project Representatives

- Charlie Robertson (MDSSC)
- Jerry Snider (MDSSC)

6.7 Case Data Sources

Bloom, Jonathan. *Requirements, Level B/C, for the Bus Loss Integrated Smart System*. October 1991.

Robertson, C. *Space Shuttle Fuel Cell Monitoring System (FCMS) (draft)*. McDonnell Douglas Space Systems Co. Houston, TX. September 1990.

Welborn, C., and C. Robertson. Multi-Threaded Procedural Reasoning in Real-Time Health Monitoring of Manned Spacecraft Systems. *Proceedings of the International Symposium on Ground Data for Spacecraft Control*. European Space Agency. Darmstadt, FRG. June 1990.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE Dec/93	3. REPORT TYPE AND DATES COVERED Technical Memorandum	
4. TITLE AND SUBTITLE Technical Memorandum		5. FUNDING NUMBERS	
6. AUTHOR(S) Jane T. Malin, NASA/JSC Debra L. Schreckenghost and Ron W. Rhoads*		8. PERFORMING ORGANIZATION REPORT NUMBERS S-752	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Johnson Space Center Houston, Texas 77058		10. SPONSORING/MONITORING AGENCY REPORT NUMBER TM 104786	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546-0001		11. SUPPLEMENTARY NOTES * The MITRE Corporation	
12a. DISTRIBUTION/AVAILABILITY STATEMENT unclassified/unlimited Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161; (703) 487-4600		12b. DISTRIBUTION CODE	
13. ABSTRACT <i>(Maximum 200 words)</i> Observations from a case study of intelligent systems are reported as part of a multi-year interdisciplinary effort to provide guidance and assistance for designers of intelligent systems and their user interfaces. A series of studies have been conducted to investigate issues in designing intelligent fault management systems in aerospace applications for effective human-computer interaction. The results of the initial study are documented in two NASA technical memoranda: TM 104738, Making Intelligent Systems Team Players: Case Studies and Design Issues, Volumes 1 and 2; and TM 104751, Making Intelligent Systems Team Players: Overview for Designers. The objective of this additional study was to broaden the investigation of human-computer interaction design issues beyond the focus on monitoring and fault detection in the initial study. The results of this second study are documented in this report, which is intended as a supplement to the original design guidance documents. These results should be of interest to designers of intelligent systems for use in real-time operations, and to researchers in the areas of human-computer interaction and artificial intelligence.			
14. SUBJECT TERMS man-computer interface; artificial intelligence; computer systems design; real time operation		15. NUMBER OF PAGES 64	16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL