REAL-TIME AUTOMATED FAILURE IDENTIFICATION IN THE CONTROL CENTER COMPLEX (CCC)

N94-23892

Sarah Kirby, Janet Lauritsen, Ginger Pack
NASA Johnson Space Center

Anhhoang Ha, Steven Jowers, Robert McNenny, The Truong
McDonnell Douglas Space Systems Company, Houston Division

James Dell
Loral Space Information Systems

ABSTRACT

This paper describes a system which will provide real-time failure management support to the Space Station Freedom program. The system's use of a simplified form of model based reasoning qualifies it as an advanced automation system. However, it differs from most such systems in that it has been designed from the outset to meet two sets of requirements.

First, it must provide a useful increment to the fault management capabilities of the Johnson Space Center (JSC) Control Center Complex (CCC) Fault Detection Management system. Second, it must satisfy CCC operational environment constraints such as cost, computer resource requirements, verification and validation, etc. The need to meet both requirement sets presents a much greater design challenge than would have been the case had functionality been the sole design consideration. This paper overviews the choice of technology, discussing aspects of that choice and the process for migrating it into the control center.

1. INTRODUCTION

This paper provides an overview of the underlying technology and design of Extended Real-Time FEAT (ERF) within the context of its migration to the CCC. Section 2 describes the fault detection and identification (FDI) problem, specifically focusing on those functions ERF now addresses and how we arrived at this focus. Section 3 overviews the constraints imposed upon the developer - both those imposed because of good software engineering and those imposed by the nature of the environment. Section 4 examines the choice of technology, the algorithms developed, and possible extensions and adjuncts to it. Section 5 overviews the solution settled upon, its current status and general comments. Section 6 finishes the paper with concluding remarks.

2. FDIR PROBLEM

2.1 The FDIR Task

From the start, this project has not sought to force a particular technology into the operations environment but rather to understand the environment and the task that the end-user must accomplish. Specifically, we addressed the fault detection, identification, and recovery (FDIR) aspects of systems monitoring and control. We sought to understand the FDIR task from the Mission Controller (MC) perspective without regard to how those tasks were to be done (i.e., by human or machine). A task analysis was performed which spanned several months and was informally documented (Ref. 1). It involved mission controllers, NASA and contractor management, and technology experts. Following this analysis, the group decided how to partition the tasks between human and machine. There were clearly some tasks which could be automated immediately and others which must be left to the MC. Still others were subjects for subsequent automation. (Since the most complex defined task is recovery, the familiar FDIR will appear henceforth as FDI.)

2.2 The Subtask To Automate

The task analysis defined three types of failures. Explicit predefined faults were defined as "those for which specific malfunction and recovery procedures are developed and validated. Typically, explicit predefined faults are time critical." Implicit predefined faults were defined as "those which are understood and for which conceptual malfunction and recovery procedures exist, but written procedures have not been developed and validated. For example, malfunction and recovery procedures normally do not address degraded systems (i.e., systems in which a failure has already occurred)." Undefined faults are those for which no *a priori* mappings exist for the signature/fault pair.

From the task analysis came a number of subtasks which the controller must perform for fault analysis, regardless of the fault type (explicit, implicit, or undefined). After examining the capabilities which the then baselined control center was to provide, we concluded that CCC baseline support existed only for explicit, predefined failures. An opportunity to expand support to include implicit failures seemed to present itself for specific failure analysis functions. These were commonality assessment among multiple alarms and/or out-of-limit conditions, identification of failures, and impact assessment. The consensus was that automating analysis of undefined faults is a research topic.

2.3 ERF's Role

Other subsystems of the CCC Fault Detection and Management (FDM) system interpret onboard Caution and Warning (C&W) codes, generate and display onboard Fault Summary Messages (FSM) and Ground Detected Onboard Fault (GDOF) messages, issue alerts and audible alarms, log FSM and GDOF messages, and pass data to the ERF subsystem for further processing. ERF augments the core FDM capability by analyzing explicit and implicit predefined faults (note: a representation of implicit faults also captures the explicit faults as a subset). ERF will be handed a problem definition (failed conditions) and after gathering other instant state information (e.g., sensors known to be within limits, degraded system state, etc.) perform an automated analysis, always under the MC's oversight and control.

ERF's capabilities can be conceptually broken into two parts; fault analysis and problem management. Fault analysis consists of Commonality Assessment, Failure Identification and Impact Assessment. The problem management functions include management of queues if multiple problems are to be worked, queuing of analysis commands, interfacing with external entities, interrupt/resume functions, etc. At any time during processing the MC can interrupt the process and enter a "hypothetical" mode of operation. The systems state information can also be used to populate the model with an initial state and the MC can then explore "what if?" scenarios based upon current, observable conditions.

3. CONSTRAINTS

ERF has been designed with the CCC as its target environment from its genesis. We have been extremely careful not to use techniques or make assumptions which would preclude its use in the CCC. We acknowledge that this may result in an approach which may be defined too conservative by those whose responsibilities are to develop and understand new technology. However, a flight-critical application by its very nature lies outside the scope of a research lab. Real-world constraints must be considered: robust interfaces to real-time data must be provided, missing and noisy sensor data must be dealt with, multiple problems must be queued, analyses must be interrupted and resumed, etc. Real-world applications need also to consider end-user constraints and implementation requirements.

3.1 End-User Constraints

MC concerns posed a number of constraints on the design of ERF. Broad user requirements are documented in (Ref. 2). ERF's user requirements are either an explicit subset of these or are derived therefrom. These include requirements for:

- Compatibility with proven real-time mission support operations
- Prompt solution response
- Positive MC control at all times
- Generic FDI capability.

Any automated system designed for real-time mission operations must support the process which human controllers currently use. MCs work in highly integrated, well trained teams. Shifting part of a team's responsibility to a computer means that the computer must play a part in the team; it cannot be a solo player (Ref. 3).

Accurate identification of the failure at hand is a prerequisite for dealing with critical situations. This means that to be useful when it is really needed, an automated FDI system must take no longer to provide a solution than would a highly experienced MC. ERF is required to identify failures and assess their impacts within 5 minutes on average, 6 in the worst case.

Just as a human team player must take orders, ERF must always be directly controllable by the MC. This means that ERF must keep the MC informed of its intentions and allow him or her to redirect it, or to halt it if necessary, at any time.

Since failure identification is in principle a generic process, ERF is required to support FDI for all Space Station onboard systems for which its basic knowledge representation scheme is appropriate.

3.2 Implementation Constraints

Any development of an application whose target is within the real, operational world needs to satisfy such real-world constraints as cost ceilings, resource allocation limits, and fast algorithm performance. Sometimes data will be noisy and/or missing. Interfaces with other CCC systems (Status and Control, database functions, the data streams, etc.) must exist. Choice of platform, development methodology and language will be made by the control center developers, not the technologists. The application and the models it uses must be susceptible to verification, validation, and maintenance.

In the case of the CCC, the constraints included the use of Ada for all new code, minimal cost, working within the constraints of the development environment for the CCC, following methodological guidelines for formal software development, and formal reviews.

4. THE TECHNOLOGY

4.1 Its Description

ERF is fundamentally built upon a bi-valued model representation. The models are causal networks of failure represented as directed graphs. They are automatically configured via telemetry and knowledge of system degradation to represent the system's observable state. These models are then operated over to infer information about commonality among annunciated out-of-limit or alarm conditions, what could have been the cause or causes of these annunciations, and to where these effects might propagate.

The Failure Environment Analysis Tool (FEAT), built by the JSC Intelligent Systems Branch, computes transitive closure for the given model, displaying the results of queries graphically. For details, see (Refs. 4-6). ERF is a layer of software on top of FEAT to extend its capabilities. Specifically, ERF sequences queries to FEAT in a manner similar the way a MC might use it to analyze either real-time data or hypothetical data. The results are displayed on FEAT's displays and/or on displays external to FEAT.

Currently, FEAT runs on both Macintosh and Unix platforms (Refs. 4-5). A companion product, the Digraph Editor (Ref. 4), provides an environment to aid in constructing failure models for FEAT, though other tools which adhere to the PICT standard can produce working digraphs for FEAT. Both products are written in C and are available for the Macintosh through COSMIC. The Unix version has just been submitted to the Space Station Freedom Program's Technical and Management Information System (TMIS) and is now available to the Space Station Program. Eventually it will be available to all through COSMIC as well. Table 1 (at the end of this paper) lists FEAT capabilities and the enhancements that ERF will provide.

4.2 Rationale

The technology was selected because it provides the required functionality and satisfies the constraints:

- It utilizes relational failure models of the systems being analyzed.
- The digraph failure models documented via a graphical representation are more easily maintainable than equivalent text representations.
- It utilizes a standard mathematical technique for computing reachability (connectivity) analyses; e.g., transitive closure.
- In addition, it utilizes heuristic knowledge about how to diagnose failures in physical systems.

From an operations perspective ERF supports real-time fault analysis and facilitates hypothetical reasoning on the part of the MC. It can be used to support training and the generation of Support Products such as malfunction procedures, Failure Modes Effects Analysis/Critical Items List (FMEA/CIL), and operations procedures. Its displays can aid in providing an explanation of results by presenting a graphically expressed chain of events. It is cost effective; it uses software engineering principles to minimize life-cycle cost and is implemented in Ada around a standard software product. It will be tested using the same tools as the rest of the CCC.

ERF allows FDI applications to be developed without the system operational failure experience heretofore thought essential for developing such systems. The digraph failure model describes, on the basis of design derived knowledge, how a system *must* fail. ERF's algorithms provide a methodology for interpreting the model based on sensor data. Since the sensor interpretation algorithms are model independent, ERF will work for any system for which a FEAT-compatible failure model is provided.

4.3 Potential

The digraph representation is both concise and capable of expressing complex patterns of failure behavior. Since ERF uses models of failure behavior, derived from knowledge of system structure, and system state information, ERF provides a simplified form of model based reasoning.

ERF can incorporate increasingly sophisticated analysis techniques within its own analysis routines and/or provide initial hypothesis generation for other external algorithms such as those contained in true model based reasoning systems.

The use of the failure model, as noted earlier, supports a number of uses in the operations community. Additionally, we envision possible uses in Recovery Planning, Intelligent Computer Aided Training (ICAT), and Design Knowledge Capture (DKC). The digraph as a knowledge source can support many applications. As operational experience is acquired, models are easy to modify and heuristic information can be easily added, either as rules or implemented in conventional, procedural code.

Digraphs and their foundation can provide much more than just simple cause/effect propagation. The digraph representation is a subset of propositional logic. Consequently, if the representation were enriched to include full propositional logic, ERF could begin using models of system behavior from which both nominal and failure behavior can be analyzed using state information.

5. ALGORITHM OVERVIEW

This section overviews only at a conceptual level the analysis algorithms which ERF implements. At the highest level, these are Commonality Assessment, Failure Identification, and Impact Assessment.

5.1 ERF Core Analysis Algorithms

The Commonality Assessment algorithm uses the digraph to determine if there are paths between components having failure indications. If there are and there are no dependencies which could impede the failure propagation, then we can assert that one of the sensors is "primary" and the others downstream of it are "secondary".

Failure Identification takes the announced conditions, retrieves additional state information and categorizes the observables in the model into one of three states; good, bad, and unknown. From this, ERF builds the initial set of possible causes. This set is then pruned using knowledge about the known good components and digraph modeling artifacts to narrow the space of possible causes. Any remaining unknown observables in the model are then presented to the MC. If additional information can be obtained, the analysis is refined. For details on an early version of this algorithm, see (Ref. 7).

The Impact Assessment function takes these possible causes and predicts their respective effects on the system. ERF will annunciate: lost redundancies, where a failure may propagate if the other leg in the redundancy is lost, and new susceptibilities for critical functions (i.e., new single and dual failure sources). Other possible subfunctions include calling out which observables to monitor for indication of a failure propagating toward a critical function.

5.2 Possible Extensions/Adjuncts

ERF has been specifically designed to facilitate growth. ERF is especially *not* viewed as the panacea for failure analysis. We readily acknowledge that the digraph bi-valued model is a low fidelity model. One possible way to increase its value is to include full propositional logic capability. Even so, it would still be bi-valued and (relatively) low in fidelity. Another possible scenario is for ERF to hand its analysis off to other algorithms. We have started working with the Thermal Control System Automation Project (TCSAP) to explore how ERF might replace some of the knowledge base portions of their system. ERF's analysis results would be used as the set of initial hypothesis for a deeper analysis by a model based reasoner in that case.

6. THE SOLUTION

6.1 Development Methodology

It is important to note the design philosophy which is being used for ERF's development. The process started with the MC defining the FDIR task without respect to its allocation between man and machine.[1] Once defined, a subset of these were selected for computer implementation; limited Fault Detection and Identification and some problem management functions (queueing, interruption, etc.). The use of FEAT and the enhancements which together comprise ERF are hence couched in a broader setting, that of an integrated FDI system. While ERF does not implement all FDI requirements, its design has been influenced by an understanding that there is a "bigger picture". Effort has been made to provide hooks for swapping underlying analysis engines and the changing of internal analysis algorithms. The use of a bi-state failure model built on partial propositional logic can only provide limited insight into a system's true behavior. Hence the need to interface with other analysis applications which use alternate model representations. It is also worth noting that ERF provides automated FDI analysis for any system which can model failure as a causal network of failure modes -it is in this sense a truly generic FDI application.

In implementing these requirements, we have sought to adhere to the guidelines and milestones for CCC development. The appropriate NASA management and contractors have been involved. Funding has been switched from one directorate to another to facilitate

1. This project had no "formal" guidelines for its development methodology at its genesis. Much was based on prior experience, common sense and fortunate timing. A recent publication (Ref. 3) is an excellent reference. We strongly recommend the reading of this memo by anyone who desires their project to receive acceptance into the operational community.

technology transfer. Mission Controllers have been involved since its inception. The net effect has been a project with MC support, management approval, and contractor acceptance. This effort has involved three JSC directorates and three NASA contractors.

6.2 Status

Today, FEAT is baselined for use in the CCC and will be delivered to the CCC Testbed in December of this year. ERF had a Preliminary Design Review in March 92 (Ref. 8) and is now baselined. Its analysis algorithms are to be available in the testbed in June of 1993 and in the CCC as part of the FDM delivery in 1994.

An informal prototype for algorithm development and detail design activities currently exists. As of October 1, 1992, it consisted of two separately running processes -the GUI and the analysis routines. While far from being the system to be delivered to the CCC, the current informal prototype demonstrates:

1. the basic analysis functions of Commonality, Failure Assessment, and Impact Assessment

2. the use of Feat's displays for analysis presentation

3. the use of additional displays built outside of Feat for results presentation

6.3 Future Directions

ERF is baselined for use in the control center, so where do we go from here? We intend to explore alternate analysis engines and representations, communication with other applications, development of more sophisticated FDI algorithms, and the recovery problem. Some of these efforts have already started, while others are still on the horizon.

7. CONCLUSION

ERF is an application to aid a Mission Controller (MC) in identifying the cause(s) and subsequent effects of observed failure symptoms in a monitored system. It is a layer of software built upon the Failure Environment Analysis Tool (FEAT) provided by JSC's Intelligent Systems Branch. The additions that the "ER" in ERF bring are; 1) automated fault identification and effects analysis algorithms which are model independent, 2) hooks for alternate model representations and alternate analysis engines, 3) interfaces to real-time data, and 4) automated problem management functions.

ERF uses advanced automation techniques. It provides automated Fault Detection and Identification (FDI) analysis for any system which can be modeled as a causal network of failure modes. As the MC identifies additional candidate functions for automation, there will be opportunity for ERF's capabilities to expand and/or for it to work with other applications. ERF has been designed to provide hooks for swapping underlying representations and analysis engines, for incorporation of more advanced analysis algorithms, and for communication with other applications.

ERF is a technology transfer project, having moved from the labs into the operational world. Its requirements are derived from a subset of the MC's FDIR task description and implemented via advanced automation techniques. It has MC support, having been designed not only with the end-user in mind but with the end-user having actively participated in the design process. Hence, its design has been influenced by an understanding of the operational environment, the MC's FDI task, and the capabilities of automation technology.

REFERENCES

1. *Functional Description* (Internal Document), April 2, 1991.

2. *Space Station Control Center User Detailed Functional Requirements.* April 1991. JSC-13192.

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

4. *Macintosh FEAT 3.4 User's Guide.* 1992. JSC's Intelligent Systems Branch. August 24, 1992. (includes the Digraph Editor User's Guide)

5. *Unix FEAT 3.4 User's Guide.* 1992. JSC's Intelligent Systems Branch. August 24, 1992.

6. Stevenson, R. et al. 1991, Failure Environment Analysis (FEAT) Tool Development Status. Presented at *AIAA Computing And Aerospace VIII Conference.* Baltimore, MD: AIAA 91-3803.

7. Clark, Colin et al. 1992. Fault Management For The Space Station Freedom Control Center. Presented at *AIAA 30th Aerospace Sciences Meeting & Exhibit.* Reno, NV: AIAA 92-0870.

8. *Space Station Control Center Extended Real-Time FEAT Subsystem Specification (Preliminary Draft).* March 1992. JSC-13416.

FEAT	ERF
Simple analysis -limited to single queries	Automated analysis using sequenced FEAT calls: Commonality Assessment -quick look commonality assessment of multiple out-of-limit and alarm conditions Find Cause -uses information about what has and has not failed Predict Effects (built using the Find Cause results) Immediate consequences Lost redundancies New system susceptibilities (Next worst failure)
Analysis uses only failure mode information (i.e., does not use what is known to be good)	Automated Analysis using both failure and non failure information
No real-time problem management capabilities	Real-time problem management functions
Manual entry of system configuration data to reflect degraded system conditions	Semi-automated entry of system configuration information for tracking degraded system conditions (e.g., The Mission Controller records system degradation via some TBD electronic method. ERF will use this information to automatically configure the system model to reflect the system's degraded condition.).
Manual entry of annunciated alarms and out-of-limit conditions	Automated entry of annunciated problem conditions
Manual retrieval of additional status information not contained in the problem announcement	Automated retrieval of additional information needed for automated analysis
	Automated entry (upon MC direction) of telemetry data reflecting current conditions when no problem has been announced
No distinction between "real" and "hypothetical" analysis	"Real" vs "Hypothetical" problem management: "Real" mode for use of real telemetry information "Hypothetical" mode for "what if" analysis Initialization of "hypothetical" mode using actual, instant conditions Provision to update "real" information from selected "hypothetical" data (note: such updates do not automatically propagate to other systems)
No hooks readily available for additional analysis algorithms, alternate on-line analysis engines, alternate model representations, etc.	Especially designed to provide hooks for: Alternate/additional analysis algorithms (well beyond transitive closure) Alternate/additional engines for performing analysis Alternate model representations (e.g., expansion to full propositional logic) Communication with other analysis applications