BMO-TR-91-26

Embedded Fault-Tolerant Computer for Mission Critical Applications

FINAL TECHNICAL REPORT

Contract No. F04704-89-C-0044

Prepared for:
Department of the Air Force
Headquarters Ballistic Systems Division
Air Force Systems Command (AFSC)
Norton Air Force Base, CA 92409-6468

Prepared by:
Gary A. Kravetz
Fail-Safe Technology Corporation
Suite 645
5757 West Century Boulevard
Los Angeles, CA 90045-6407

August 1, 1991

Document No. FST91-281-2

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION IS UNLIMITED.

This report has been reviewed and is approved for publication.

James M Tynan, Maj, USAF
Name and Grade of Project Officer
or Scientist

Name and Grade Supervisor
Col, USAF

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
01 AUG 1991

3. REPORT TYPE AND DATES COVERED
Final Nov 89 - 01 AUG 91

4. TITLE AND SUBTITLE
Embedded Fault-Tolerant Computer for Mission Critical Applications

5. FUNDING NUMBERS
C: F04704-89-C-0044

6. AUTHOR(S)
Gary A. Kravetz

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Fail-Safe Technology Corporation
5757 West Century Boulevard
Los Angeles, CA 90045-6407

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
BALLISTIC MISSILE ORGANIZATION/SE
NORTON AFB, CA 92409-6468

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
BMO-TR-91-26

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
A. Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Fail-Safe Technology successfully designed and tested a low cost fault-tolerant computer for mission critical applications. This computer is software and hardware compatible with the IBM PC and is a viable product for both commercial and government markets. The report defines the research into the requirements for the computer and the architectural tradeoffs and the field test results.

14. SUBJECT TERMS
Fault-Tolerant, Fail-Safe, Mission Critical Computer, Embedded Computer, Ruggedized Computer

15. NUMBER OF PAGES

16. PRICE CODE

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

A7CA2

Table of Contents

List of Figures

List of Tables

1. scope

1.1 Identification

This document is the Final Report for contract No. F04704-89-C-0044 entitled *Embedded Fault-Tolerant Computer for Mission-Critical Applications* awarded to Fail-Safe Technology Corporation (FST) by the Ballistic Missile Office (BMO) under the Air Force Systems Command (AFSC) as a Phase II Small Business Innovation Research (SBIR) contract.

1.2 Purpose

The objective of this effort is to develop an Embedded Fault-Tolerant Computer (EFTC) using a combination of off-the-shelf and custom hardware and software components. This prototype will then be analyzed in order to determine the concept feasibility.

1.3 Introduction

This report summarizes the main activities and results of the EFTC development as it has progressed through Phase II. It describes: the development of the EFTC system from evolution through hardware and software design (Section 2); brassboard implementation of hardware and software (Section 3); and the successful demonstration of generic capabilities as well as applications (Section 4). In addition, summaries of various ways in which the system can be enhanced illustrate the flexibility of the design (Section 5).

2. System Development

In this section we describe the way in which the EFTC system was developed in terms of the underlying software processes and the hardware architecture.

2.1 Evolution

The first task of this SBIR effort was to determine more detailed requirements for the EFTC. We started by discussing existing and future projects with Norton BMO personnel. After a few briefings to the Rail Garrison and Small Missile project offices at Norton, we were directed to vendor of services and systems to Norton. We talked to Rockwell, TRW, GTE, and Ford Aerospace about the requirements for fault-tolerant computers in ICBM systems.

The conclusion was that no great needs could be identified for fault-tolerance in the missiles because of short mission duration and existing safety systems. Also, no new ICBM designs are currently funded or expected in the near future.

Possible requirements for fault-tolerant computers were identified for the ICBM ground support equipment (GSE), test equipment, and command, control and communication (C^3) systems. The most common need we found for fault-tolerant computers was for communication control computers using a PC-class workstation. Solutions not requiring a fault-tolerant computer had been developed for the existing equipment, but a fault-tolerant computer would have been used if it were available.

In looking at the needs of future programs, no new equipment designs were identified. Programs like Rail Garrison and small ICBMs were based on existing designs. The conclusion of this study was that there was a need for fault-tolerant PC-class computers in new ICBM GSE, C^3, and test equipment, but no new equipment was planned for the foreseeable future.

Personnel at BMO suggested we talk to AFSC for possible applications. The Advanced Program Office at AFSC indicated that few new satellite programs were planned, and existing programs had all developed solutions to meet their reliability requirement. The need for fault-tolerant computers in new programs was primarily in the GSE. In looking at the typical requirements for GSE, a PC-class workstation using Intel 386 or 486 processors would meet most requirements.

In addition to the requirements of BMO and AFSC, FST looked at other government and commercial application customers. We determined that there was a need for a fault-tolerant computer at a lower price than the existing commercial fault-tolerant computers such as Tandem and Stratus. In addition, users wanted hardware that would run low-cost software developed for the IBM PC-compatible hardware platforms.

In view of those discussions, the decision was made in conjunction with BMO to develop the EFTC as a fault-tolerant IBM PC-compatible platform that functions as a network-based file server.

2.2 Concept and Approach

Developing a fault-tolerant computer is not an easy task. The design is constrained by many factors that results in several tradeoffs while developing and meeting a set of requirements. In order to make this task tractable, we have approached it by using a well-established and systematic design paradigm for fault-tolerant systems; the next section outlines the steps in this paradigm.

2.2.1 Fault-tolerance Design Paradigm

Given a set of system requirements that define the services to be delivered and the service boundaries at which service delivery will take place, the key steps of the paradigm are as follows:

1. The dependability goals of the system are specified in terms of reliability, availability, maintainability, safety, and so forth. This requires three steps.

   (a) First, the classes of hardware and software faults that are to be tolerated over the life of the system are explicitly identified. Fault classes are chosen such that faults that elicit the same error syndrome are grouped into the same class, thereby reducing the scope of the effort.

   (b) Second, quantitative goals for the dependability of system services are specified.

   (c) Third, the methods for evaluating the dependability actually attained by the system are specified in detail.

2. The system is partitioned into subsystems (hardware, software, communication, interfaces) for implementation, taking into account both performance and fault-tolerance.

3. Error detection and fault diagnosis algorithms for every subsystem are selected. The choices of error detection and fault diagnosis techniques are guided by the dependability goals. We ascertain that all relevant fault classes are detectable, and that the probability of timely detection is adequate.

4. State recovery and fault removal techniques are devised that are invoked by the fault signals from fault detection algorithms. Their goal is to return the system to some level of proper operation or to shut it down safely. Fault signal invoked recovery is classified into three classes during the design process:

   (a) recovery to original performance (fail-operational);

   (b) recovery to degraded performance (fail-soft);

   (c) execution of safe shutdown (fail-safe).

   A fourth class of recovery algorithms that does not depend on a fault signal, but maintains original performance by the use of concurrently active protective-redundancy is also considered (masking).

5. Subsystem fault-tolerance is integrated into the overall system.

6. An evaluation of the fault-tolerance of the design and its impact on performance is then performed via a combination of analytic and simulation modeling.

7. A refinement of the design is then carried out. If the initial evaluation demonstrates that the various hardware and software subsystems fail to meet the primary dependability specification, or if there are

unequal contributions to the overall system dependability, steps 2 through 6 are repeated. The goal of this refinement step is to balance the protection provided to each subsystem so that the dependability goal is achieved without a single dominating contributor to nondependability, and at the lowest cost of additional resources.

2.3 System Specification

The specification of the EFTC hardware and software was accomplished by using the design paradigm of Section 2.2.1 in conjunction with a structured development methodology [HaP88]. The design paradigm defines the key steps of the design and the structured development methodology provides a stylized mechanism for developing and documenting each step. Our approach is to first define fault-tolerance requirements for the system based on its dependability goals, and then develop detailed specifications via a System Specification Model (SSM).

2.3.1 Basic System Description

The following top-level system description is provided here as a basis for better understanding the system development process and models that follow.

The EFTC provides file-server (FS) functions for workstations in a local-area network (LAN) environment. The baseline hardwarc architecture is depicted in Figure 1. Workstations on the LAN may use the file server as an information repository or to perform other network-wide services such as a centralized printer. Workstations interface with the file-server by sending file-server requests across the LAN, and-receive responses in the form of data or control messages from the file-server.

The file-server runs the Novell Netware operating system, and workstations interface to it via Novell-compatible client modules that execute locally.

2.3.2 Fault-Tolerance Requirements Definition

In this section we establish the fault-tolerance requirements for the EFTC system.

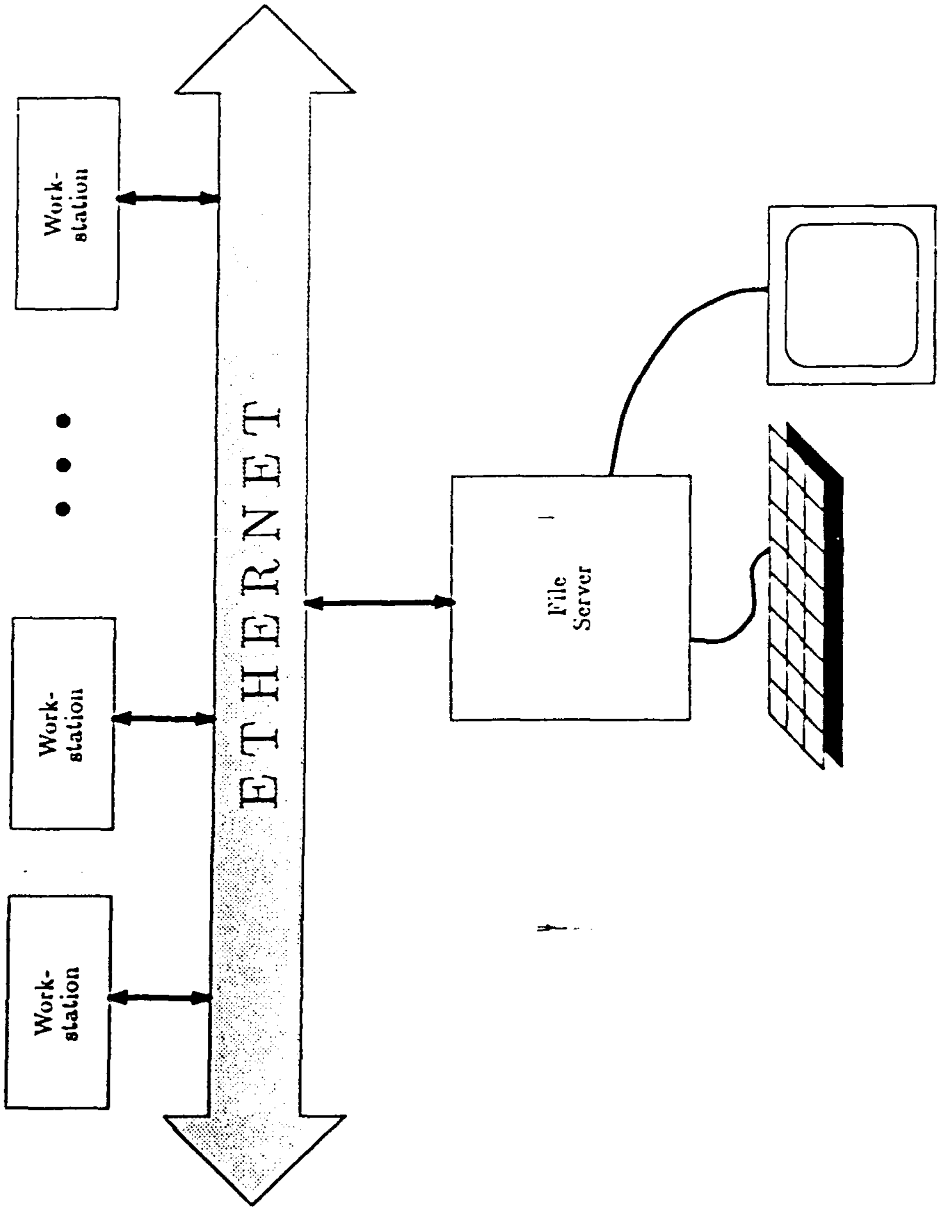

Figure 1: Baseline File-Server Architecture

2.3.2.1 Fault Set Definition. A comprehensive fault model of the system was constructed. This model lists all faults that are to be mitigated and their attributes. To limit the amount of faults that need to be considered, faults that generate the same error or failure response were grouped into the same class; the objective being to "cover" as many faults as possible with as few mechanisms as possible.

The basic fault set was constructed by assuming that the "service boundary" encapsulates the entire system; the service boundary is an imaginary interface at which service is delivered to a user. Users in this case are workstations on the network, or an operator that interfaces directly with the system via a system console. The faults in the set therefore account for all possible faults within the system that can lead to a failure from a user's perspective. This resulted in the following list:

- Any permanent fault, or any transient or intermittent fault with a duration greater than 5 seconds, that results in loss of system power.
- Any permanent fault that results in loss of the CPU.
- Any permanent fault, or any transient or intermittent fault with a duration greater than 5 seconds, that results in partial or total loss of the contents of the RAM.
- Any permanent fault, or any transient or intermittent fault with a duration greater than 5 seconds, that results in loss of the ethernet controller.
- Any permanent fault, or any transient or intermittent fault with a duration greater than 5 seconds, that results in loss of the SCSI disk controller.
- Any permanent fault, or any transient or intermittent fault with a duration greater than 5 seconds, that results in loss of critical information on a hard-disk that directly supports file-server operations.

The decomposition of this basic fault set into subclasses depends on the next step in the paradigm, which is to partition the system into subsystems based on system-level fault-tolerance and other requirements.

2.3.2.2 Dependability Goals. The fault-tolerant design of the EFTC depends on its dependability goals. Since it is going to be used in an environment where manual repair is possible, we felt that an availability requirement was more imp_.tant than a reliability requirement. Whereas reliability defines the probability of correct service delivery for a specified "mission time," availability defines the long-term or steady-state probability of correct service delivery. In other words, we felt it was more important that the system be operational a large portion of time, even though it may suffer infrequent outages due to failures. The availability requirement defines the percentage of time the system must be operational.

For this effort, steady-state availability, A_{ss}, was defined as:

$$A_{ss} = \frac{\text{MTTF}}{\text{MTTR} + \text{MTTF}}$$

where MTTF $= 1/\lambda$ and MTTR $= 1/\mu$, and λ and μ represent constant failure and repair rates, respectively. MTTF is defined as the Mean Time to Failure, and MTTR the Mean Time to Repair. Given this definition, A_{ss} was determined in the following way: Of all the faults in the basic fault set (see Section 2.3.2.1), our experience with PC-AT-class machines in a file-server configuration as depicted in Figure 1 is that hard-disk failures is the dominant failure class with an approximate MTTF of 5000 hours. Our experience also is that after a hard-disk failure, the MTTR is anywhere from 2 to 8 hours with a mean of about 4 hours, depending on parts availability. If spare disks are on-hand, then the repair time is about 2 hours to replace the disk and perform a restoration from backup volumes. Given an MTTF of 5000 hours and an MTTR of 2 hours, A_{ss} for the baseline non-redundant configuration is 0.996 (99.6%). Our goal was to improve this availability considerably. Since a hard-disk is a prepackaged unit, there is no way to improve its MTTF directly. However, effective MTTF of the disk can be improved via replication of the hard-disk unit. For example, if a second (physically identical) hard-disk is provided, and if it has a probability of failure independent of the first hard-disk, then the effective MTTF is the sum of the MTTFs of both hard-disks. This assumes, of course, that both hard-disks are configured so that they represent a single logical hard-disk. The second way to improve the availability is to reduce the MTTR. We felt that it was possible to reduce the MTTR from two hours to at most a few minutes through the judicious application of fault-tolerance. For example, if the MTTF is reduced to 2500 hours—to be conservative—and the MTTR to 5 minutes, then A_{ss} improves to 0.99997. On the basis of this analysis, our goal for A_{ss} was set to > 0.9999 (this roughly corresponds to at most 17.5 minutes of unavailability per calendar year of continuous operation). An associated maintainability goal was to achieve an MTTR of < 5 minutes.

2.3.2.3 Dependability Evaluation. The design paradigm requires that the methods used to evaluate the dependability goals specified in Section 2.3.2.2 be specified at this point. The plan for meeting the maintainability goal—MTTR < 5 minutes—was relatively simple: we simply measure the maximum time required to recover from all faults in the fault set. Evaluating availability is more difficult. Evaluating availability by measuring the ratio of uptime to total time is infeasible since the total time must be very long so that a statistically meaningful number of system failures can occur (this time is estimated to be about 10000 hours—roughly one year of continuous operation). The alternatives were simulation modeling and analytic modeling. Our choice was to use analytic modeling of the final hardware configuration since the time to develop and run suitable simulation models is beyond the scope of this effort.

2.3.3 Detection and Recovery Algorithms

The next step of the design paradigm requires that detection and recovery algorithms be chosen. Although this is an iterative process that cannot be completed until other steps of the paradigm are complete—such as the fault set, it was an opportunity to make some high-level decisions about the way in which detection and recovery will be handled in the EFTC.

One design constraint was the need to reduce the amount of custom hardware and software necessary to implement the EFTC, while retaining compatibility with standards—*de facto* or otherwise—developed for the class of machines chosen. While it is possible to implement fault-tolerance anywhere from the system to the component level, the cost and complexity increases the closer we get to the component level. Our first choice, therefore, was to determine if the dependability requirements could be met by applying fault-tolerance at the system level. If this is not possible, then the next choice would be fault-tolerance at the level of individual subsystems, and so on.

System-level fault-tolerance requires that replication be used at the system level; i.e., the entire system is replicated. Error detection algorithms then monitor and detect errors at the system boundary, and error recovery involves the dynamic manipulation of entire systems. In the case of the baseline system (see Figure 1), the "system" that is to be replicated is the box labeled "File Server" on the diagram. This system is a 386-based PC-compatible configured to operate as a file server using the Novell Netware operating system. In our first attempt at applying fault-tolerance, this system was duplicated in such a way that each duplicate can perform the necessary file-server functions.

There were several choices of error detection algorithms for a duplexed system configuration. One choice was to operate both systems concurrently, loosely or tightly synchronized, and use a comparison algorithm to determine when the two systems disagree; diagnostics could then be run on both systems to determine which one was faulty. Another choice was to design each system to be self-checking so that it is capable of detecting its own faults and removing itself from the rest of the system at that time. Our experience was that such self-checking systems require redundancy at the very lowest levels of the system in order for it to reliably detect internal faults—this is what we were trying to avoid from the outset. A third approach is to treat one of the systems as the "active" system, and the other as a "standby" system. The standby system does not normally perform file server operations, but instead monitors the state of the active system. An error condition in the active system detected by the standby system results in activation of an error recovery procedure. Our choice of error detection mechanism was the latter one since it requires no, or relatively little, custom hardware and software beyond what is already in the system. In this configuration, our error detection algorithm monitors the state of the active system by the standby system using the builtin mechanisms of the Novell Netware operating system and the ethernet hardware. The basis of the detection algorithm is conceptually simple: any fault that results in the inability of the file-server to provide a basic class of service (such as accessing a file for read and/or write) results in invocation of an error recovery algorithm. All faults in the basic fault set are "covered" by this algorithm. The resulting fault-tolerant system configuration is depicted in Figure 2.

Given the choice of error detection algorithm, the recovery algorithm was chosen to effect recovery subject to the dependability requirements (primarily the recovery latency). In our case, the choice was straightforward. When an error is detected in the active system, the active system is removed from the configuration, and the standby system is initialized and brought online as the active file server. When the failed system is repaired, it assumes the role of the standby system.

The detailed design and implementation of these algorithms are presented in later sections of this report.

2.3.4 System Specification Model

Given a baseline fault-tolerant system configuration, we then constructed a SSM to define the basic data and control flows necessary to achieve fault-tolerant file-server functionality, and an appropriate detailed system architecture. The resulting SSM comprises a System Requirements Model (SRM) and a System Architecture Model (SAM). The relationship between these models is depicted in Figure 3. Our development models were not automated and, due to the relatively small amount of custom hardware and software that was used in the brassboard design, only the essential parts of the development methodology are included in this report.

2.3.4.1 Requirements Model. The requirements model abstractly defines the hardware, software, and other requirements of the EFTC. Its principal tools are flow diagrams—data flow diagrams (DFDs) and control flow diagrams (CFDs). Figure 4 depicts the data context for the EFTC via a data context diagram. The corresponding control flow context is depicted in Figure 5. The data context diagram depicts the data relationship between the EFTC and its surrounding environment. The circle in the diagram represents a "process"—this one representing the EFTC system,

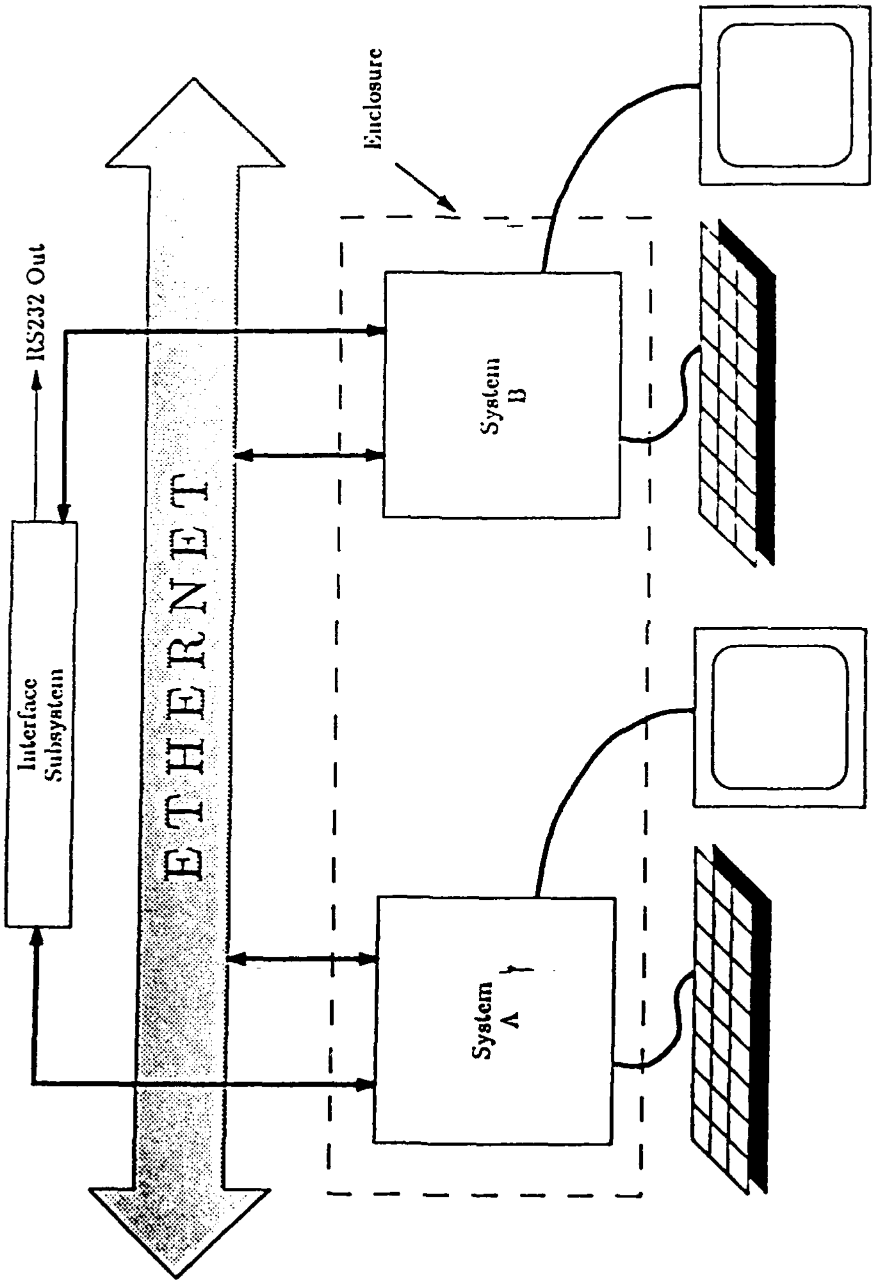

Figure 2: Baseline EFTC Architecture

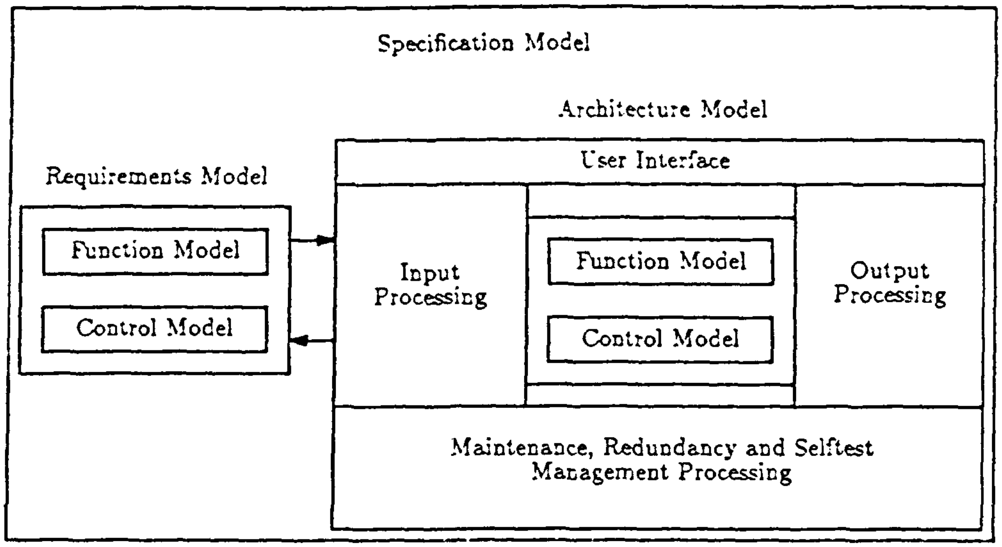

Figure 3: EFTC System Development Process

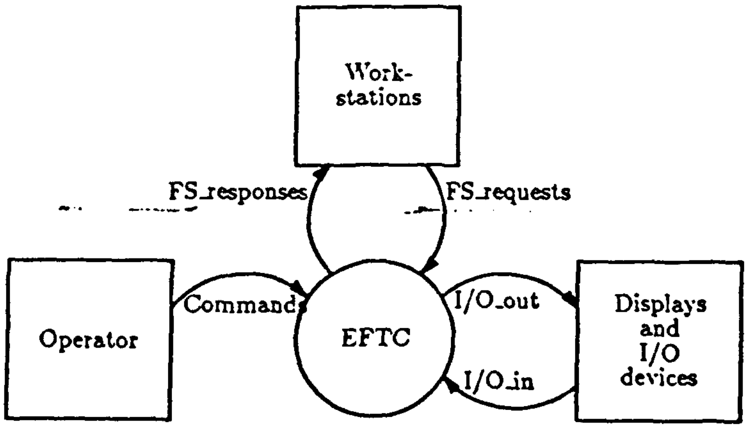

Figure 4: Data Context Diagram

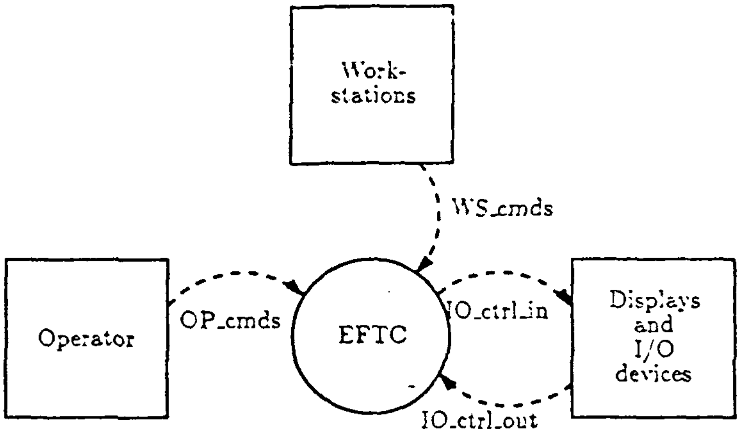

Figure 5: Control Context Diagram

boxes represent static entities in the environment, and directed arcs represent the flow of data. For example, a workstations send and receive data from the EFTC in the form of file server requests and responses, respectively; the labels on the arcs indicate the class of data flowing along that arc. The data context diagram is the highest level of activity in the system; it is the top level of a tree of DFD's, each of which provide successive levels of refinement in data flow within the EFTC system. The control context diagram is identical to the data context diagram except that the directed arcs indicate the flow of control between the EFTC and its environment. For example, workstations may send discrete control signals to the EFTC to initiate or abort file server actions, or an operator may issue control commands to the EFTC. Like the data context diagram, the control context diagram is the top-level of a tree of such diagrams, each of which provide successive levels of refinement in control flow within the EFTC.

The first level of decomposition in the requirements model results in the top-level DFD and CFD diagrams depicted in Figures 6 and 7, respectively. The DFD, and the associated CFD, show three processes. Since we were not modifying the basic system functions of the EFTC baseline software, these are lumped into the single process named 'System'. The file-server functions were broken out into a separate process since it was necessary to establish the data relationships between fault-tolerance functions 'Fault

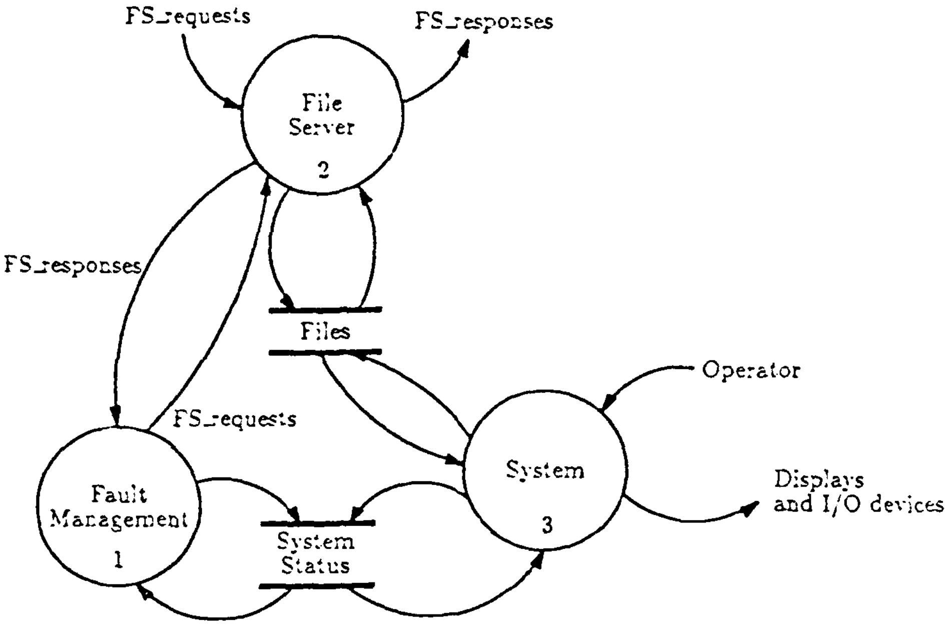

Figure 6: DFD 0: Fault-Tolerant File-Server Operation

Management' and file-server functions 'File Server'. The CFD shows the control relationship between these processes.

The second level of decomposition in the requirements model results in DFD and CFD diagrams depicted in Figures 8 through 11, respectively. They show decompositions of process 1 'Fault Management' and process 2 'File server'. Process 1 is by far the most important since it requires the most customization. (In fact, all custom software in the EFTC belongs to this process.)

2.3.4.2 Architecture Model. The architecture model abstractly defines the configuration of physical modules that perform all the required data and control processing. The requirements from the requirements model were mapped into an architecture model taking all design constraints into account. These constraints included all the requirements defined in the Configuration Item Development Specification, FST document No. FST91-281-1, CDRL 002A2.

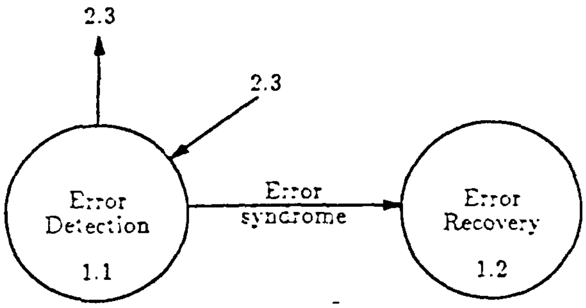

Figure 8: DFD 1: Fault Management Process

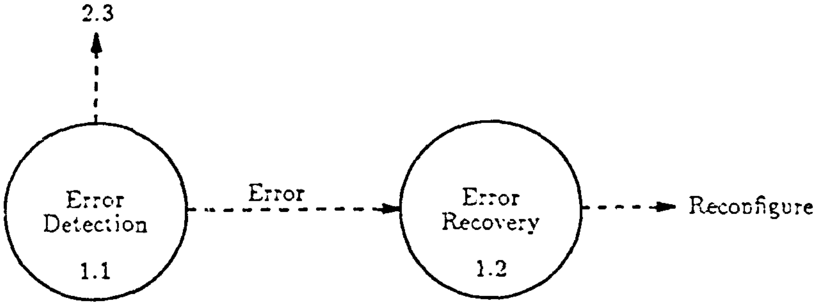

Figure 9: CFD 1: Fault Management Process

- A local 80 Mbyte hard-disk drive and a 3.5 inch floppy disk drive accessed via a disk controller card.
- A SCSI disk controller that accesses a pair of 80 Mbyte hard-disk drives.
- An ethernet controller card that facilitates communication over the external ethernet cable.
- A serial I/O interface card that provides several RS232-compatible ports.
- A card that contains the CPU, RAM, and display-driver subsystems.
- A PC compatible keyboard.
- A color monitor.

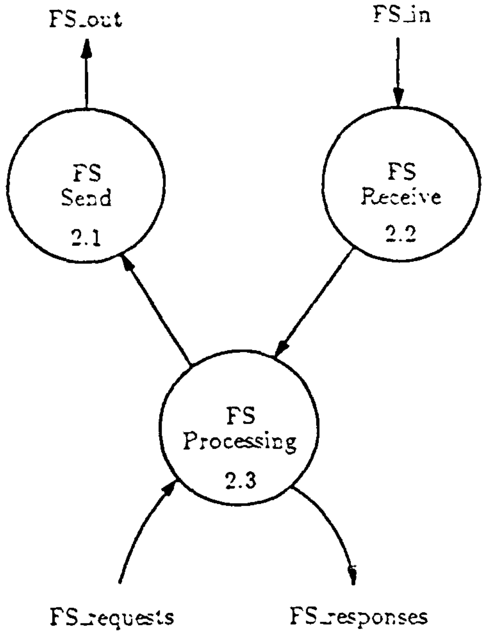

Figure 10: DFD 2: File-Server Process

The backup system has an extra RS232 I/O card that is used to provide an error signal when an error is detected in the primary system.

The "Switch Box" is a custom-designed subsystem that provides physical switching of RS232 I/O lines and SCSI disk I/O lines between the two computers. It is independently powered from the computers.

3.2 Software

3.2.1 Software System Architecture

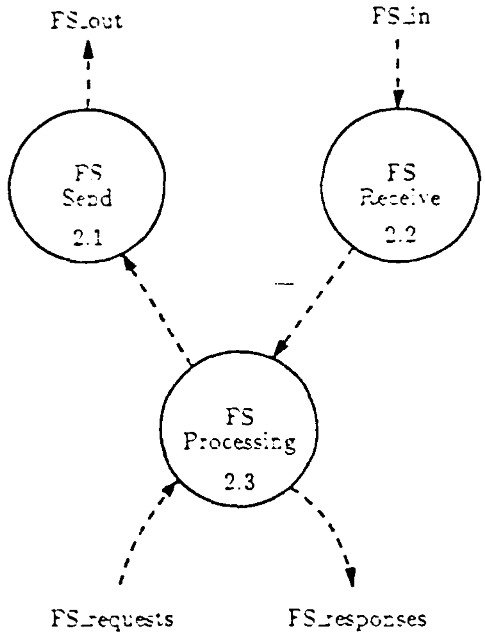

Figure 11: CFD 2: File-Server Process

4. Application Studies

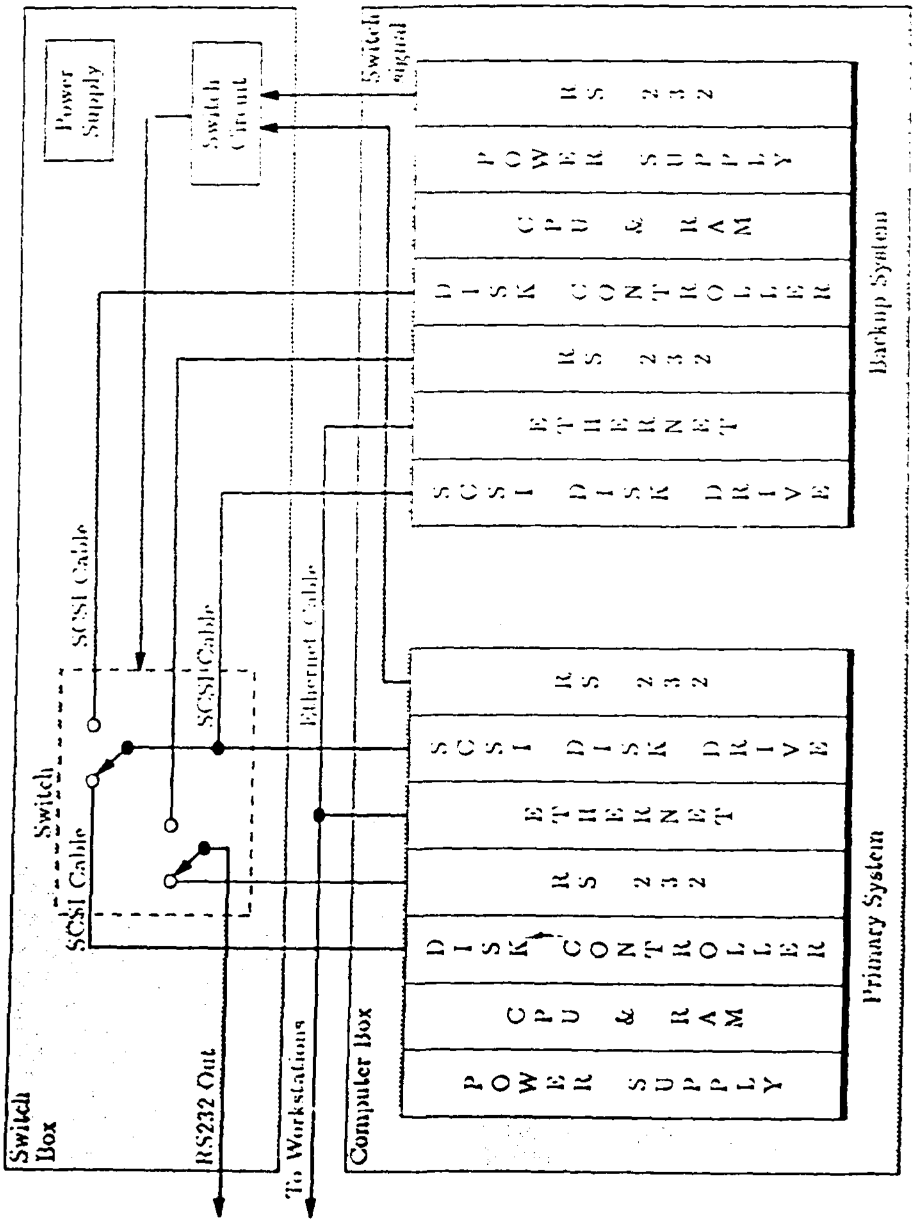

Figure 12: EFTC Hardware Architecture

5. System Enhancement Studies

In order to enhance the useability of the EFTC, a study was performed to determine how the architecture could be adapted to perform as a file and terminal server in a UNIX environment. This section discusses hardware and software changes necessary to support a UNIX environment.

5.1 Hardware Configuration

We began with the baseline EFTC hardware depicted in Figure 2. This configuration was extended to include additional hardware needed to support a typical UNIX environment. The revised baseline architecture is depicted in Figure 13. The Interface Subsystem comprises a mirrored-disk subsystem and an ethernet multiplexer-demultiplexer (EMD), as well as the physical switching hardware. This switching hardware switches the RS-232 inputs/outputs of the EMD as well as the SCSI disk cable from one computer to the other. Additional hardware required and not shown are EMDs at the workstations; these are not part of the EFTC.

5.2 Software Configuration

The operating system selected for each PC compatible is UNIX-VR4 (AT&T UNIX System V, Release 4) because of its position as a *de facto* industry standard. Additionally, each PC compatible is configured with the following software:

- A Fault-Tolerance System Manager (FTSM) module for system-level error detection and error recovery.
- A SCSI driver.
- An ethernet driver.
- Network File System (NFS) manager. (Basic NFS is supplied as part of the operating system.)
- Terminal I/O port Manager (TIM).
- Network Manager.

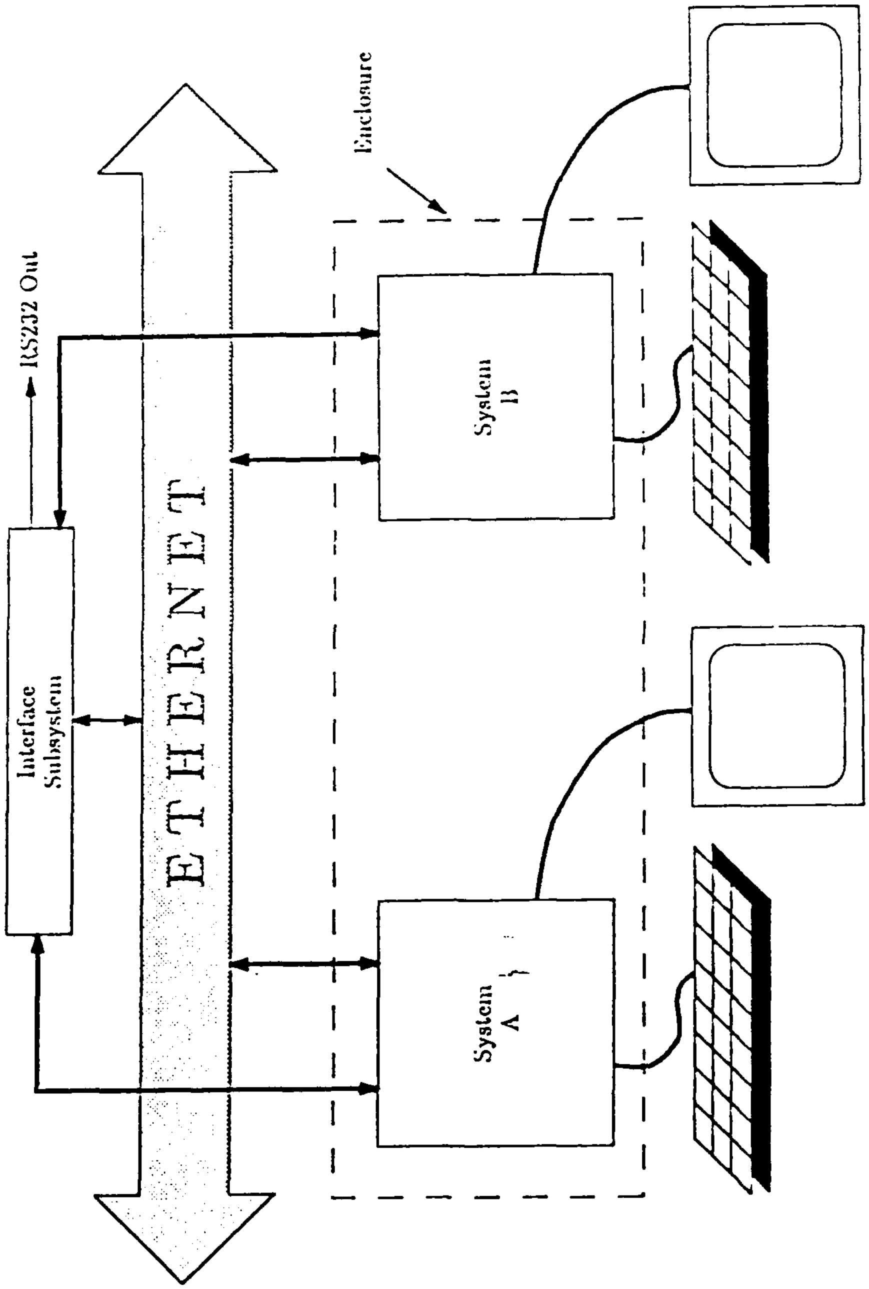

Figure 13: UNIX Baseline EFTC Architecture

Table 1: EFTC Failure and Operating States

Failure States		Operational States		Comment
A	B	A	B	
F	F	S	S	System failed
F	F	S	A	System failed
F	F	A	S	System failed
F	F	A	A	System failed
F	O	S	S	Transient
F	O	S	A	System OK (degraded)
F	O	A	S	Transient
F	O	A	A	System OK (degraded)
O	F	S	S	Transient
O	F	S	A	Transient
O	F	A	S	System OK (degraded)
O	F	A	A	System OK (degraded)
O	O	S	S	Transient
O	O	S	A	System OK
O	O	A	S	System OK
O	O	A	A	Impossible

5.3 Basic Operating Principles

The EFTC would achieve fault-tolerance through system-level error detection and spare-switching. One PC compatible is designated as "active," and the other as a "spare." Table 1 depicts the failure and operational states of the system, where 'F', 'O', 'A', and 'S' mean failed, operational, active, and standby, respectively. When a fault is detected in the active system by the spare system, spare switching occurs and the spare system becomes the active system; more details of this operation are provided in the following paragraphs.

5.3.1 Initialization

At system startup, both systems boot UNIX from their local hard-disks. After a time period sufficient for both systems to complete their boot processes, or upon operator intervention from either system, an initialization dialogue commences between the two FTSMs to determine which system will be active. The *activity* protocol is designed in such a way that, if both systems boot correctly, system 'A' will become the active system; otherwise, the system that boots correctly will become active. Once "mastership" has been established, system operations may begin.

5.3.2 Error Detection

The EFTC uses system-level error detection; i.e., the spare system is used to detect errors in the active system. More precisely, the FTSM in the spare system communicates and coordinates with the FTSM in the active system to detect errors and perform recovery actions.

Errors in the active system are detected using the normal error detection mechanisms of the system; i.e., the errors detected by the hardware and/or the errors detected by the system software. Each error is assigned an *error class*. The FTSM in the spare system monitors these errors and, based on their severity, initiates system recovery actions. The FTSM in the spare system also detects an error *by omission*; i.e., if the active system fails to respond to periodic queries from the FTSM in the spare system, the active system is deemed to have failed.

5.3.3 Error Recovery

The EFTC uses system-level recovery. The standby system will, as part of the recovery actions, assume mastership and become the active system. The terminals and external hard-disks will be physically switched to the new active system, terminal ports and file systems will be logically connected to the active system, and system operations resumed.

In order to support Level 2 transparency in file server operations (Level 2 transparency means that users will observe a system outage while error recovery takes place; they are not able to continue the current session, but there will be no loss of data due to the failure), file server "transactions" are monitored by the spare system (before the failure of the active system). Upon spare-switching, any transactions that had not committed by the failed system will be redone by the newly active system. Terminal ports are simply reconnected to the active system—users are required to relogin.

5.3.4 File Server Operation

Clients external to the EFTC system may access and use its file server functionality. File server functions are accessed via remote file server commands across the ethernet. The EFTC currently implements the NFS protocol for remote file operations. (Other file server protocols can be optionally supported.)

5.3.5 Terminal Server Operation

Users external to the EFTC system may login remotely via the terminal server functionality. The EFTC provides a number of serial ports for this purpose. During system recovery, all serial ports are physically switched from one side of the system to the other; any users currently logged in via one of these serial ports will have their sessions terminated—they must login again when recovery is completed.

6. Summary Test Report

All of the systems tested by Fail-Safe Technology (FST) were configured on pairs of personal computers (two computers in one box). Initial tests were run on the Diversified Technology single box PC-compatible computer system. Netware 286 network software was installed and failure modes tested. A Netware 286 system was configured on Unisys computers (supplied by NASA) and returned to them for evaluation.

As NASA's procurement activities were operating on a large multi-server network with hundreds of users, their test results were as valid and reliable as the in-house test performed at Fail-Safe Technology. The NASA tests have been conducted over an 8 month period to ensure comprehensiveness. Fail-Safe Technology then installed Netware 386 network software on an IBM compatible "clone" style computer tower 486 system.

The IBM clone system was never made to work reliably due to a few voids in true compatibility, which is evident in some brands sold in the marketplace. FST installed the same software on an NCR desktop computer system. This system worked well and reflected problem-free operation when switched without open files.

6.1. Tests with Diversified Technology Single Box PC-compatible Computer System

Initial tests were run on this equipment with DOS programs. Software was written and coded. This operating system add-on program transferred keystrokes from each PC into a keystroke buffer of the second machine. Hardware was designed, developed and fabricated to transfer an RS232 communications serial interface input into both machines within the single box and selected the output of the active primary computer. This design had limited success. Some off-the-shelf programs (such as "Windows") did not use the keyboard buffer, therefore, failed to function with this technique. Other programs (e.g., PC Base IV) would omit occasional keystrokes and go out of synchronization. Results of primary market research conducted on logical potential applications exposed interest from only a limited number of prospective customers for fault-tolerant DOS applications. There were few DOS applications identified in the market for critical operational functions. The few prospects that were interested in fault-tolerant DOS (such as security monitoring applications) had implemented special programs (custom software). This is logical.

6.2. NASA System (Unisys computers with Novell 286 network software)

Efforts were concentrated on creating a fault-tolerant system that would operate on off-the-shelf Novell network programs. FST installed Novell 286 on a hardware system consisting of two Unisys computers sent to us by NASA-Houston and returned for installation and test operations.

FST created a software program to monitor the primary network file server from a backup secondary machine. When the backup was unable to access a file on the primary it switched the primary SC SI disk to itself and re-booted as the Novell file server. The system performed without problems in FST's laboratory and was shipped to NASA for installation and operation in a real-world environment as a Beta test site. NASA encountered the following problems over time while the system was in use:

a. The backup would sometimes log into and monitor another server rather than its own primary. This was due to the enormous numbers of alternate servers operating on the system. This was corrected by FST with a software modification and enhancement.

b. The backup would switch over when long (size and time) file transfers were tying up the network. FST resolved this problem with a software change permitting a lengthier time for the back-up to receive the correct information from the primary computer.

c. The backup would sometimes not complete its booting without operator input when the files were open when the switch occurred. This problem was partially resolved, but, requires additional software enhancements to improve the reliability in this situation. Novell's program must be modified to accomplish this.

6.3 Tests with Other Hardware

FST installed Netware 286 network software on both of the Diversified Systems computers with hardware designed and fabricated by FST to troubleshoot the problems identified by NASA as shown above. FST solved problems a and b with software modifications and enhancements. Problem c was not totally solved. Coordination with Novell will be required to alter their software slightly and accommodate resolution of this problem. It was decided to convert and upgrade the system to Netware 386, which might not manifest this problem. The Novell Netware 386 would probably be demanded by all future customers, anyway.

FST installed Netware 386 on two IBM compatible "clone" computer towers with switching hardware and software designed, developed and fabricated by FST. Data was corrupted on almost every simulated failure during testing. The clones were returned to the vendors as a result. FST then installed Netware 386 on NCR 3445 computers for further testing. The switchover worked as well on NCR as the Netware 286 performed on the Diversified System.

6.4 Conclusion

The FST fault-tolerant adapter hardware performed very well on the off-the-shelf microcomputers most accepted in the marketplace, converting them to fault-tolerant systems. Problems were experienced with a single "no name" brand clone, which was discovered not to really be 100% IBM compatible due to design nuances in the hardware.

FST's hardware has been tested to the point where it is considered to be ready for production.

7. References

[HaP88] D. J. Hatley and L. A. Pirbhai, *Strategies for Real-Time System Specification*. San Francisco, CA, Dorset House Publishing, 1988.

BMO-TR-91-26

INITIAL DISTRIBUTION LIST
for
FINAL TECHNICAL REPORT
for
EMBEDDED FAULT-TOLERANT COMPUTER FOR
MISSION CRITICAL APPLICATIONS
Contract No. F04704-89-C-0044

Ballistic Misssile Organization/SE
Norton Air Force Base, California 92409-6468

James M. Tynan, Major USAF

Sam Crow

R. G. McNeal

Fail-Safe Technology Corporation

Gary A. Kravetz

Kenneth B. Smernoff

Timothy R. Robinson

Dr. Michael W. Sievers