

RL-TR-94-65
Final Technical Report
May 1994

AD-A281 251



ADAPTIVE FAULT TOLERANCE

GE Aerospace Advanced Technology Laboratories

Sponsored by
Ballistic Missile Defense Organization

DTIC
ELECTE
JUL 06 1994
S G D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Ballistic Missile Defense Organization or the U.S. Government.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

94-20500



SOP

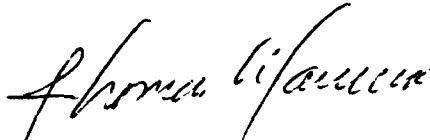
DTIC QUALITY INSPECTED 3

94 7 5 128

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-65 has been reviewed and is approved for publication.

APPROVED:



THOMAS F. LAWRENCE
Project Engineer

FOR THE COMMANDER



JOHN A. GRANIEFF
Chief Scientist for C3

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

ADAPTIVE FAULT TOLERANCE

L. T. Armstrong

Contractor: GE Aerospace Advanced Technology Laboratories
Contract Number: F30602-89-C-0182
Effective Date of Contract: 28 August 1989
Contract Expiration Date: 1 December 1991
Short Title of Work: Adaptive Fault Tolerance

Period of Work Covered: Sep 89 - Nov 91

Principal Investigator: Len Armstrong
Phone: (609) 866-6253

RL Project Engineer: Thomas F. Lawrence
Phone: (315) 330-2805

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input checked="" type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution unlimited.

This research was supported by the Ballistic Missile Defense Organization of the Department of Defense and was monitored by Thomas F. Lawrence, RL (C3AB), 525 Brooks Rd, Griffiss AFB NY 13441-4505 under Contract F30602-89-C-0182.

DTIC QUALITY INSPECTED 3

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 May 1994		3. REPORT TYPE AND DATES COVERED Final Sep 89 - Nov 91	
4. TITLE AND SUBTITLE ADAPTIVE FAULT TOLERANCE				5. FUNDING NUMBERS C - F30602-89-C-0182 PE - 63223C PR - 2304 TA - 01 WU - 02	
6. AUTHOR(S) L. T. Armstrong				8. PERFORMING ORGANIZATION REPORT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) GE Aerospace Advanced Technology Laboratories Bldg 145 Moorestown Corporate Center Moorestown NJ 08057				10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-94-65	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Ballistic Missile Defense Organization 7100 Defense Pentagon Wash DC 20301-7100				Rome Laboratory (C3AB) 525 Brooks Rd Griffiss AFB NY 13441-4505	
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Thomas F. Lawrence/C3AB/(315) 330-2805					
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.				12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The objective of the Adaptive Fault Tolerance program is to provide large complex distributed military systems with greater degrees of survivability, and graceful degradation than is currently available. Most research on these systems to date has focused on the management of static threat and environmental conditions. However, many military Battle Management/Command, Control, Communication, and Intelligence systems exist not in a static but in a highly dynamic environment. The dynamics occur along several dimensions such as alternate modes of operation, changing threat type or threat rate, loss of system resources such as communication links or processing assets, and changing network topology and asset configuration. Using static fault-tolerance approaches in these systems is inappropriate because system requirements may change as a result of changes along one or more dimensions in the dynamic operating environment. Furthermore, designing a system for worst-case situations in every dimension of conceivable threat is cost prohibitive. An adaptive approach to fault management enables the system to dynamically tailor its fault tolerance/survivability mechanisms to best deal with a changing environment and to apply limited system assets appropriately. This effort was not completed due to lack of funds. This interim report represents the only output from the effort and will be published as a final report.					
14. SUBJECT TERMS Distributed Systems, Fault Tolerance, Adaptivity, Resource Management, Command and Control Systems				15. NUMBER OF PAGES 56	
				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	

1. Introduction and Overview

This report documents the work performed on the Adaptive Fault Tolerance (AFT) contract (contract number F30602-89-C-0182), sponsored by the U.S. Air Force's Rome Laboratory, Command and Control Directorate, Computer Systems Technology Branch (C3AB). Thomas F. Lawrence was AFT's Contracting Officer, Technical Representative. The work documented in this report was performed during the 27 month period from September, 1989 until November, 1991.

The objective of the AFT program is to provide large complex distributed military systems with greater degrees of survivability, availability, and graceful degradation than is currently available.

Many systems currently address requirements for high availability and reliability through the use of various fault-tolerant strategies to detect and recover from potential problem areas. Most research on these systems to date has focused on the management of static threat and environmental conditions. However, many military Battle Management/Command, Control, Communication, and Intelligence (BM/C3I) systems, such as the Strategic Defense Initiative (SDI) Strategic Defense System (SDS), exist not in a static but in a highly dynamic environment. The dynamics occur along several dimensions such as alternate modes of operation, changing threat type or threat rate, loss of system resources such as communication links or processing assets, and changing network topology and asset configuration. Because the continued effectiveness of these systems is essential to our national security, fault tolerance, survivability, and continued operation are critical attributes that must be provided.

Using static fault-tolerance approaches in these systems is inappropriate because system requirements may change as a result of changes along one or more dimensions in the dynamic operating environment. Furthermore, designing a system for worst-case situations in every dimension of conceivable threat is cost prohibitive. An adaptive approach to fault management enables the system to dynamically tailor its fault tolerance/survivability mechanisms to best deal with a changing environment and to apply limited system assets appropriately.

This report is divided into three major sections. The remainder of Section 1, *Overview and Introduction*, provides background and logistics of the AFT program. Specifically, program organization, team members, and milestones achieved are discussed.

Section 2, *Research*, details a multitude of topics related directly to the theoretical basis and vision of AFT. Included are subsections on definition and focus of the AFT concept; AFT system architecture description; the Adaptive Behavior Manager, a method for insertion of AFT into a large system; a taxonomy of faults in large complex distributed systems; traditional or static fault tolerance techniques which serve as a basis for potential adaptation;

adaptations which can be made within an AFT system; and a notional example which demonstrates a practical application of the presented theory.

Finally, Section 3, *Demonstration*, discusses the AFT simulation system which was developed as a concept proof of many of the theoretical issues presented in Section 2.

1.1. Program Organization

To achieve the AFT program goal of providing greater degrees of availability, survivability, and graceful degradation to large complex military systems, the program was organized into two phases: research and concept demonstration.

The cumulative result of the research phase is the specification of a formal structure for AFT concepts which serves as a cornerstone for further AFT technology development and insertion and as a common framework for interested researchers and developers.

The research phase consisted of a wide variety of individual tasks which collectively form the resultant formal structure. These tasks include concept definition, system architecture specification, adaptive behavior management, fault taxonomy development, fault tolerance technique classification, investigation of potential adaptations, and the development of notional examples which serve to continually shape the AFT framework and associated requirements.

In the concept demonstration phase many of the abstract concepts developed in the research phase were applied to a simulated BM/C3I application modelled loosely on the Monitor Function of the Strategic Defense Initiative (SDI) Command Center Element. The resulting demonstration system allows researchers to interactively inject faults into the simulation system, and observe the effect on the system's fault tolerance strategies.

1.2. Team Members and Technology Thrusts

The AFT program is sponsored by the U.S. Air Force's Rome Laboratory. The technical efforts of the GE AFT team are directed in Rome Laboratory by Thomas F. Lawrence.

The structure of AFT is founded in variety of technologies. Hence, building the proper team was essential for the AFT program to be successful. Each member of the AFT team brings a unique set of technical skills and practical problem knowledge to the development of this new technology.

The AFT team consists of three organizations from GE (the Advanced Technology Laboratories, the Strategic Systems Department, and the Corporate Research and Development Center), the Concurrent Computer Corporation, and Dr. Kane H. Kim from the University of California at Irvine. GE's Advanced Technology Laboratories (ATL) served as prime contractor and program lead. Figure 1-1 shows an organization chart of AFT team members and the technology/development areas under their responsibility.

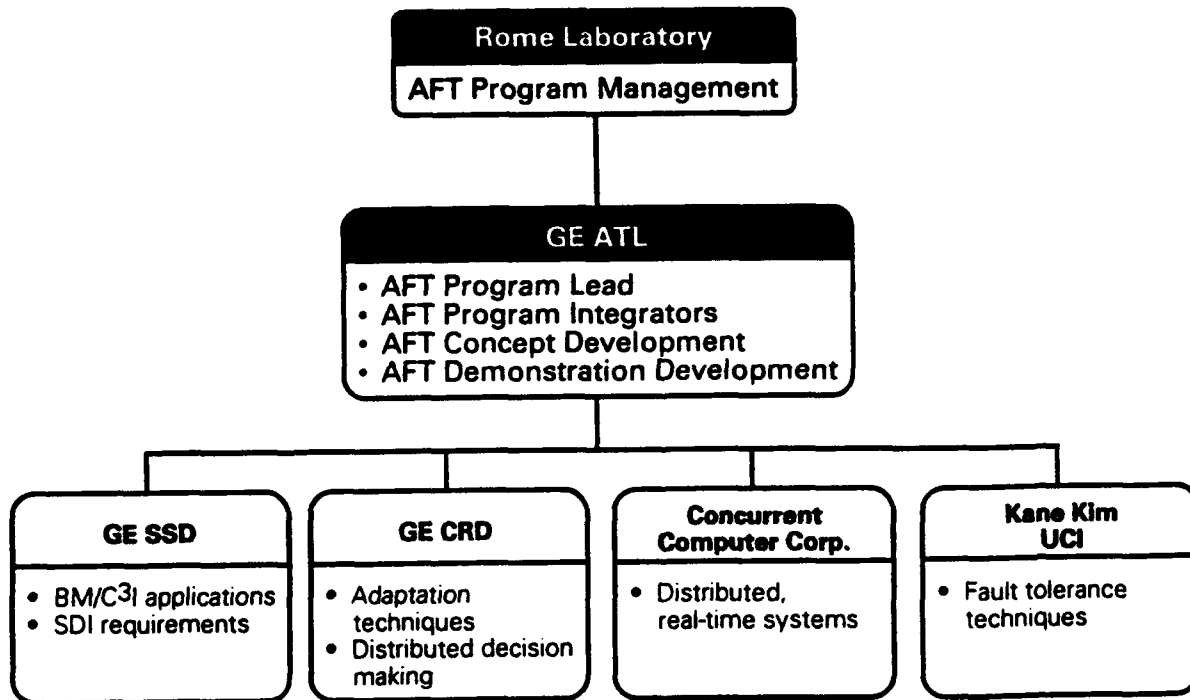


Figure 1-1
AFT Program Organization

The primary technologies which serve as the foundation for AFT are: distributed real-time computer systems, traditional fault tolerance and fault management techniques, distributed decision making, and an understanding of BM/C3I applications and the SDI system in particular.

The AFT program is centered in the Digital Processing Laboratory of GE ATL. Along with serving as the prime contractor, ATL's technical responsibilities within the program include program integration, AFT concept development, and concept demonstration implementation.

Expert knowledge of the current state-of-the-art in fault tolerance and fault management is a critical component in the development of AFT. The GE team was complemented in this area with the talents of Dr. Kane H. Kim of the University of California at Irvine. Dr. Kim is a respected and highly published researcher in the field of fault tolerance, especially as applied to large distributed military systems.

As lead contractors of the SDI Integration Effort, GE's Strategic Systems Department (SSD) provides unique expertise in the design, and requirements of large military BM/C3I applications, including the SDI system.

GE's Corporate Research and Development Center (CRD) is responsible for adaptation strategies that are needed to properly implement AFT. CRD's experience in the fields of distributed decision making, rule-based scheduling, and case-based planning provided the proper core for adaptive behavior management technology growth.

Finally, Concurrent Computer Corporation rounds out the GE team with expertise in the areas of real-time distributed systems and state-of-the-art real-time scheduling techniques.

1.3. Milestones

Including program award, the GE team reached four significant milestone since program commencement.

First, in September, 1989, the AFT program was awarded from Rome Laboratory to the GE team.

Second, in September, 1990, the paper "Adaptive Fault Tolerance: Issues and Approaches" was published in the *Proceedings of the Second IEEE Computer Society's Workshop on Future Trends of Distributed Computing*. Tom Lawrence and Kane Kim authored this paper.

Third, in September, 1991, a completed version of the AFT interim demonstration was shown to the NATO Study Group on Distributed Systems during a Study Group Meeting at Rome Laboratory.

Fourth, also in September, 1991, the paper "Adaptive Fault Tolerance" was published in the *Proceedings of the 1991 System's Design Synthesis Technology Workshop*, sponsored by the Naval Surface Warfare Center, and the Office of Naval Technology. Leonard T. Armstrong and Thomas F. Lawrence were the authors.

2. Research

The goal of AFT program research phase was to establish a formal structure of abstract or theoretical AFT concepts to act as a cornerstone for further AFT technology development and insertion and serve as a common framework for researchers and developers of future AFT systems.

To achieve this goal the research phase was divided into 7 individual tasks: concept definition, system architecture specification, adaptive behavior management, fault taxonomy development, fault tolerance technique classification, investigation of potential adaptations, and the development of notional examples.

Although the tasks were separate, each tasks had some interrelationship with other tasks. These tasks are described in greater detail in sections 2.1 through 2.7, respectively.

2.1. AFT Concept

The driving focus behind the development of AFT was to provide higher degrees of availability, survivability, and graceful degradation than is currently available in nonadaptive systems. This concept applies to large, complex military systems that operate in a highly dynamic environment where mission, mode of operation (e.g., peace time, alert, and battle), and threats to the system change.

One of the consequences of experiencing faults is a loss of resources. Because there is a finite amount of resources in the system, the resource manager must decide to what purpose the limited resources will be put. As faults lower available system resources, re-optimization of resource usage becomes necessary. Similarly, for systems that operate in highly dynamic environments, the requirements placed on these resources may change frequently. For example, increasing the system's processing load or changing the system's objective function based on current mission or mode of operation will precipitate similar re-optimization.

More specifically, Adaptive Fault Tolerance is defined as the ability to change the system's fault-management mechanisms or modify their parameters in run time to accommodate changes in the system's resource management objective function, and fault profile, with respect to available resources.

The system's fault profile refers to all aggregates associated with faults that a system can experience during operation. Primary focal points include fault type, fault rate, and probability of fault occurrence.

The system's resource management objective function consists of a set of the system's highest level requirements domains: performance, functionality, and consistency.

Performance refers to the time characteristics (e.g., response/delay, throughput, hard deadline) specified for a task or set of tasks within a system. Explicit run-time resource management to meet these performance specifications makes the system a real-time system.

Functionality refers to the different types of activities the system can perform (e.g., tracking, correlation, diagnostics, housekeeping). It also refers to the importance of a given task within each of these activities. The greater the number of simultaneous activities and tasks, the more complex the function.

Consistency refers to the specification of an acceptable/unacceptable (anomalous) system state that is used by the fault-management mechanisms in the system. Consistency is composed of several subcategories including mutual/non-mutual and internal/external.

Mutual consistency exists when several distributed entities must maintain some relationship, as in the case of replicated distributed data in which updates need

to be synchronized to maintain mutual consistency among the several replicated copies. Non-mutual consistency exists when only one copy of a resource exists and consistency criterion is established for its internal state (e.g., concurrency control for multiple users of shared data).

For both mutual and non-mutual consistency, internal and external consistency are defined. Internal consistency refers to a static consistency criterion specification, where a minimum amount of application semantics are required for the fault management to function (e.g., backward error recovery or read/write serializability for concurrency control). External consistency refers to a situation where the consistency specification can change to provide adaptivity. In this case, the semantics of the application become critical (e.g., forward error recovery or decreasing the precision of the processing to produce an approximate solution in less time). Additional fault-management mechanisms above and beyond those needed to support internal consistency are necessary if external consistency is specified.

Therefore, given a finite set of resources and the objective function consisting of functionality, performance and consistency, the resource manager in the system must attempt to optimize the benefit provided by the system. In a highly dynamic system, the objective function, the load, and the available resources can change.

Because trade-offs are made among performance, consistency, and functionality, one of these requirements cannot, in general, be increased or decreased without affecting the others. Changes along one of the dynamic dimensions of the system's operating environment affect the system's performance, functionality, and consistency requirements, which in turn affect adaptations of the system's fault-tolerance/survivability mechanisms, over which there is some parameterized control.

2.2. System Architecture

Traditional fault-tolerance schemes maintain internal consistency through static fault-tolerance techniques in a fairly straightforward fashion; faults (inconsistencies in the system state) are detected and then fault recovery mechanisms are used to bring the system back into a consistent state.

Adaptive Fault Tolerance builds upon this strategy. First, both the internal and external components of the system state are evaluated. An inconsistent system state is then defined to be an undesirable match between the external and internal states of a system. The external state places dynamic requirements on the internal state of the system. Changes in the external state of a system may force inconsistencies in the internal state. In other words, changes in the system's operating environment force adjustment of system requirements that are no longer efficiently met by the system's current internal state. Alternatively, changes in the internal state, e.g., uncontrollable loss of resources, may also cause inconsistencies between the internal and external states.

Once the inconsistencies have been detected, the system requirements must be evaluated in the context of the changes in the operating environment. Continuous requirements evaluation is most clearly understood in terms of performance, functionality, and consistency.

If the requirements have changed, then it must be decided if the system's internal state still efficiently meets the new requirements. If the current internal state does not efficiently meet the new requirements set, then a new internal state must be determined to better match the given external state. Ideally, the new desired internal state, when matched with the given external state, will cause the system to align more effectively with the new requirement set forth as a result of the state change.

If it is determined that a new internal state is required, a method to achieve the new state must be formulated. The formulated method provides specific actions to be taken to alter the system's current internal state into the more desirable internal state.

Once a method to achieve the new internal state is formulated, adjustments are made to the system to bring its current internal state to the recently determined desired internal state. Adjustments to the internal state to avoid anticipated threats to system survivability may take on various forms. A few examples are dynamic task (re)allocation; load shedding; a priori scheduling to avoid conflicts that could produce an anomalous state; replicated processing (masking); changes in computational precision; approximation techniques for processing, data, or communications; and increasing/decreasing fault-tolerance overhead.

As the system's resource management objective profile or fault profile changes, an adaptive system maintains continuity of operation by selecting a new set of the above mechanisms to handle the new fault/requirement situation.

2.3. Adaptive Behavior Management

Adaptive fault tolerance is achieved in a system through the use of Adaptive Behavior Managers (AB Managers). The specific purpose of the AB Manager is to closely model the behavior characteristics described in the previous section in order to implement AFT technology within a real system. An AB Manager is responsible for controlling all aspects of AFT behavior in some component of a large distributed system. The AB Manager considers the system component over which it has AFT-control as its domain of responsibility or internal state. An AB Manager interacts with other AB Managers in the system to achieve cooperative control of survivability adaptations.

AB Managers may have different granularities of responsibility, thus mapping effectively onto different abstract levels of a large distributed system. For example, there may be one large-grained AB Manager that is responsible for adaptation control of an entire system. The same system may also have a number of finer grained AB Managers that are responsible for adaptation control of particular subsystems. The decision-making strategy, types of decisions made, and actions taken within each of the different grained AB Managers are also tailored toward the abstract system view contained within each AB Manager.

Primary responsibilities of the AB Manager at any level of abstraction are state assessment, requirements analysis, desired state generation, and policy generation. The AB Managers are supported by knowledge bases of system-constraint information.

First, an AB Manager must assess the state of the world as it understands it. This involves both internal and external state assessment. State assessment can be achieved through traditional methods, such as database query, system observation functions, user-input, through more sophisticated methods such as knowledge-based situation assessment and automated reasoning, or by communication with other AB Managers. Coarser grained AB Managers can provide external state information to finer-grained AB Managers because the domain of influence of a coarser grained AB Manager would be larger (and more abstract) than the domain of influence of a finer grained AB Manager. Conversely, a finer grained AB Manager can provide internal state information to a coarser grained AB Manager that has a scope that encompasses (as a minimum) the finer grained AB Manager. This capability is provided as it is assumed that coarser grained abstraction levels have a loose hierarchical domain over finer grained abstractions.

External state assessment and the ability to communicate with the external world are at the very heart of AFT because AFT is linked to a system's need to maintain external consistency. External state assessment provides the ability to determine the requirements of external consistency and to alter fault-tolerance mechanisms accordingly.

After state assessment, the AB Manager must perform a revised requirements analysis. Because requirements are linked closely to the application, much of the requirements analysis must come from application-supplied sources.

If, after requirements analysis, it is determined that the system requirements have changed, the AB Manager must decide whether the current internal state can efficiently meet the updated system requirements. If the current internal state is insufficient, then a new internal state must be determined as it applies to the domain of responsibility over which the AB Manager has control.

Once an AB Manager decides that the internal state should be adjusted, a more specific formulation of how that adjustment should be achieved is needed. The forms that internal state adjustments may take vary greatly depending upon the abstraction level of an AB Manager.

The level of internal state abstraction used within an AB Manager is proportional to its granularity. Coarse grained AB Managers may implement internal state changes as commands to finer grained AB Managers, global resource reallocation, or dynamic task migration. Finer-grained AB Managers may implement internal state changes as extreme as task termination, local network reconfiguration, application algorithm alteration, fault tolerance technique internal parameter adjustment, dynamic reconfiguration of fault tolerance techniques, or computational precision adjustments.

2.4. Taxonomy of Faults

To properly evaluate the effects of changes in a system's fault profile as related to successfully meeting system requirements, one first needs to clearly understand the types of faults that can occur in large distributed systems. To accomplish this in the AFT program, a taxonomy of large distributed system faults was developed. This taxonomy serves as the basis for categorization of fault management techniques (cf. Section 2.5).

At the top level of the taxonomy faults are described according to their attributes in three separate classes: Environment Features, Fault Features, and Auxiliary Features.

Environment Features are those attributes of a fault that are directly related to the design and operation of a particular system. The Environment Features class has two subclasses: System Features, and Deadline Features. The System Features subclass is used to specify the locality of a fault's influence. Values which can be specified in the System Features subclass follow the traditional three-tiered computer network model: *single computing station*, *local area network*, and *wide area network*.

The Deadline Features subclass is used to specify the real-time implications of experiencing a fault. Values which can be specified in the Deadline Features subclass are: *soft deadline*, and *hard deadline*.

Fault Features are those attributes of a fault that are directly related to the error that is propagated, regardless of the system environment or application. The Fault Features class has three subclasses: Output Behavior Features, Multiplicity Features, and Component Features.

The Output Behavior Features subclass is used to specify a fault according to its visible propagated effect. Values which can be specified in the Output Behavior Features subclass are: *crash fault*, *omission fault*, and *erroneous fault*. A crash fault is experienced when a module has a complete, unrecoverable crash failure. An omission fault is experienced when an event does not occur within a proper time frame. Examples of omission faults are: data being unavailable when needed; a node being unable to communicate with another node on a network within a reasonable period of time; and a processor not finishing a real-time computation within a hard deadline. An erroneous fault is experienced when an event occurs incorrectly. Examples of erroneous faults are: data being decidably incorrect through use of an acceptance test; and incorrect computation being performed as a result of operator input error.

The Multiplicity Features subclass is used to specify the magnitude of occurrence of a fault within a system. Values which can be specified in the Multiplicity Features subclass are: *single*, *multiple staggered*, and *temporary blackout*. A single magnitude of occurrence implies that the effect of a fault is localized to one component of the system with little or no probability of effect in other components. (The granularity of the affected component is specified in

the System Features subclass). A multiple staggered magnitude of occurrence implies that an experienced fault has affected more than one component of the system. Temporary blackout is a special case of magnitude of fault occurrence that can arguably be placed under other subclasses instead. Temporary blackout implies that a fault has briefly taken down an entire component of the system, but that the component was able to return to operation within a short time period. After a temporary blackout the affected component may return to the system in a "hot," "warm," or "cold" state.

The Component Features subclass is used to specify the hardware and/or software entity that was affected by a particular fault. This subclass attempts to generalize computer system components to a small limited set for ease of use and understanding, yet still be specific enough so that fault tolerance techniques can be adequately classified. Values which can be specified in the Component Features subclass are: *hardware processor node*, *hardware storage node*, *software node*, *broadcast channel communication link*, and *point-to-point channel communication link*.

The third and final top level class of fault attributes is Auxiliary Features. Auxiliary Features are those attributes of a fault that are not properly classified within the Environment Features of Fault Features classes. The Auxiliary Features class has two subclasses: Occurrence Rate Features, and Predictability Features. The Occurrence Rate Features subclass is used to specify the frequency of experiencing a particular fault. Values which can be specified in the Occurrence Rate Features subclass are: *high rate*, and *low rate*.

The Predictability Features subclass is used to specify if a particular fault had been expected to occur. This subclass is useful in potentially hostile military environments where external situation (e.g., battle mode) may dictate that a system's physical components could be in danger of attack. Values which can be specified in the Predictability Features subclass are: *with warning*, and *without warning*.

Having first defined the seven subclasses of attributes which are used to describe faults, a fault can then be defined as a 7-tuple:

$$F = (S, D, B, M, C, R, P)$$

where **S** is the fault's System Features attribute, **D** is the Deadline Features attribute, **B** is the Output Behavior Features attribute, **M** is the Multiplicity Features attribute, **C** is the Component Features attribute, **R** is the Occurrence Rate Feature attribute, and **P** is the Predictability Features attribute.

For example, a typical fault could be described as:

$F = ($ **S** => Computing Station
D => Soft Deadline
B => Erroneous Fault
M => Single
C => Software Node
R => Low Rate
P => Without Warning $).$

This fault might specify an error which is experienced by an engineer running a piece of scientific software on his personal workstation. The error might have been propagated due to a design flaw in a piece of software which does not handle one special (and rare) input case properly.

2.5. Fault Tolerance Techniques

Having developed a suitable taxonomy of distributed system faults, the next step was to classify fault tolerance techniques according to the fault types which they address.

To date, an initial classification has been done for six distinct fault tolerance techniques: Periodic Diagnosis with Monitor Nodes, Recovery Blocks, Distributed Recovery Blocks, N-Modular Redundancy, Abort-Propagating Transactions, and Compensating Transactions. These techniques are well documented in the literature.

Periodic Diagnosis with Monitor Nodes (PD/MN) is primarily a fault detection technique, where a small set of nodes (potentially one) monitor the system by performing periodic diagnosis on the other nodes. The purpose is to shorten fault latency (the period during which faults are present).

Recovery Blocks (RB) is a scheme for structuring and prioritizing multiple algorithms aiming for the same or similar computations ("try-blocks") together with a reasonableness check ("acceptance test") and a rollback and retry operation. RB facilitates backward recovery, software fault tolerance, and the structuring of resilient atomic actions.

Distributed Recovery Blocks (DRB) is a distributed version of the RB technique. In DRB, each try-block is executed on a separate processor. After execution of the applicable try-block, each processor executes the acceptance test for determination of success or failure. Watchdog timers are also used to avoid hard real-time failures, by timing-out (i.e., failing) try-blocks which are not completed within a specified time limit. DRB enables efficient real-time forward recovery from hardware malfunction or software defects.

The N-Modular Redundancy (N-MOD) scheme uses three or more distributed copies of a computation (analogous to an RB/DRB try-block) and a majority voting mechanism to determine correct result. This scheme facilitates forward recovery, and thus, has predictable real-time advantages similar to DRB.

Both Abort-Propagating Transactions (APTRANS) and Compensating Transactions (CTRANS) are variations of transaction processing. Transaction processing is used to implement a series of actions as an atomic element where either all the effects of a transaction are permanent or none of the effects remain past the life of the transaction. APTRANS and CTRANS are transaction schemes intended specifically for lengthy transactions when data locking is not always convenient for extended periods of time. In these two techniques, a primary transaction allows other transactions to read data which has already been modified during the execution of the primary transaction, and before the primary transaction has fully committed.

In the APTRANS scheme, if the primary transaction is aborted, aborting actions must be taken on all other transactions which have read any altered data before

the decision to abort occurred. Hence, an abort in one transaction could have a propagating effect, causing aborts in other transactions as well.

In the CTRANS scheme, if the primary transaction is aborted, application-dependent compensating actions are used to bring the non-primary transactions back into a correct (or at least acceptable) state. Although forward recovery is achieved through the use of compensating actions for the non-primary transactions, the application domains for which compensation actions exist is somewhat limited.

Using these six techniques and the taxonomy of faults as initial building blocks, a table was developed to classify the techniques according to the fault types to which they apply. The goal was to create a table that was easily manageable and visually understandable. Two problems prevented this: the overall large number of elements in the table (dictated by the total number of fault types), and the large number of table dimensions (dictated by the number of subclasses in the fault taxonomy).

Given that there are 3 choices of System Features, 2 choices of Deadline Features, 3 choices of Output Behavior Features, 3 choices of Multiplicity Features, 5 choices of Component Features, 2 choices of Occurrence Rate Features, and 2 choices of Predictability Features, the total number of possible fault types is:

$$3 \times 2 \times 3 \times 3 \times 5 \times 2 \times 2 = 1080.$$

This number was considered high for initial manageability. Therefore, the Auxiliary Features class (subclasses: Occurrence Rate Features and Predictability Features) was eliminated from the initial table. After elimination, the total number of possible fault types that the table addresses is:

$$3 \times 2 \times 3 \times 3 \times 5 = 270.$$

Since faults in the taxonomy are categorized by a seven-way cross product (i.e., a 7-tuple) of finite information, the ideal table for classification of fault tolerance techniques is seven dimensional – one dimension for each field of the 7-tuple. However, a seven dimension table is difficult to visualize for textual information retrieval. The decision to eliminate the Auxiliary Features class reduced the number of fields in the relevant fault categorization to 5. This made the problem less severe, but the resulting table would still be unreadable at 5 dimensions.

This problem was overcome by observing that the number of possible choices for many fields in the remaining 5-tuple is low. Therefore, the number of dimensions in the table could be reduced by combining fields in the 5-tuple. By combining System Features (3 possible elements) and Deadline Features (2 possible elements) subclasses and the Output Behavior Features (3 possible elements) and Multiplicity Features (3 possible elements) subclasses, the resulting table could be reduced to three dimensions of 6, 9, and 5 elements each.

To further optimize the readability of this table, it was noted that a finite three dimensional image could be displayed as a series of two-dimensional images. Hence, the table was ultimately formulated as six 9x5 two-dimensional tables.

A final table optimization was made after noting that a temporary blackout effected all system components simultaneously. Therefore, a fault tolerance technique that is applicable to temporary blackouts should apply equally to all system components. This allowed the rows of the table that represented a temporary blackout to span all the Component Features with only one entry.

Appendix 1, shows the resulting Fault Categorization Table which currently classifies the six fault tolerance techniques described previously according to their applicable fault types.

2.6. Potential for Adaptations

Initially, it may seem that the potential for adaptation comes solely from the Fault Categorization Table. However, this table provides only a small amount of the total information that is required. Additional information is needed for AFT to be effectively and efficiently used by modern BM/C³I systems. This is especially true if adaptations are among distinct fault tolerance techniques.

In general, there are five types of adaptations that can be made in an AFT system:

- Type 1: Transition between fault tolerance techniques.
- Type 2: Adjustment to internal parameters of fault tolerance techniques.
- Type 3: Transition between application algorithms.
- Type 4: Adjustment to internal parameters of application algorithms.
- Type 5: Dynamic resource management.

Type 1 adaptations are generally considered the most difficult to implement due to the direct influence many modern fault tolerance techniques have on the overall design of a fault tolerant system. An important goal of Type 1 adaptations is the capability to dynamically transition between distinct fault tolerance techniques with little or no burden placed on the application. The black box approach was taken to achieve this goal. That is, sets of techniques which are candidates for Type 1 adaptations were required to have the majority of their internal hidden from the application, with only a common set of application interfaces made public.

After some research, three initial candidates for Type 1 adaptations were identified:

1. Adjustment between Recovery Blocks and Distributed Recovery Blocks.
2. Adjustment between Abort-Propagating Transactions and Compensating Transactions.
3. Adjustment between Programmer Transparent Coordination with Obedient Receivers and Programmer Transparent Coordination with Adaptive Receivers.¹

Type 2 adaptations are primarily intended to shift resource consumption used by fault tolerance techniques. These adaptations stress dynamic control over internal parameters of fault tolerance techniques, leaving the application in a static state. Candidate techniques for Type 2 adaptations include:

¹ Programmer Transparent Coordination with Obedient Receivers and Programmer Transparent Coordination with Adaptive Receivers are not described here, although these techniques were studied during the AFT program.

1. Altering the level of redundancy in N-Modular Redundancy or Distributed Recovery Blocks.

2. Altering the frequency of Periodic Diagnosis or Checkpointing.

Type 3 adaptations require that the application has some knowledge about the use of AFT within the system. Type 3 adaptations require a list of alternative algorithm segments from an application, and some information as to the costs and benefits of using each algorithm alternative.

An example of a Type 3 adaptation is switching from a Connection Machine version of a target identification algorithm to an alternate (but slower and less informative) version which operates on a single CPU Sun-4. This may be required if the site which houses the system's Connection Machine has fallen due to hostile attack.

Type 4 adaptations are similar to Type 2 adaptations in that they are primarily intended to shift resource consumption in a particular component of a system. The difference is that in Type 4 adaptations there is dynamic control over the application, while fault tolerance techniques are left in a static state.

An example of a Type 4 adaptations is computing with imprecise results, as researched by Jane Liu at the University of Illinois.

In Type 5 adaptations, dynamic system resource management techniques are used to benefit the system's fault management capabilities. Common Type 5 adaptations include, load balancing to avoid real-time omission errors, dynamic processes shedding and reallocation (i.e., fail-over operation) in the event of component failure.

2.7. Notional Example

A notional example was developed to better understand the practical application of many AFT theoretical concepts prior to completion of the demonstration system. Although modeled after a small portion of a large-scale military command and control application, the example can be adapted to a variety of other systems that operate in a highly dynamic environment and require advanced fault-tolerance concepts.

Section 2.7.1 provides an overview of the architecture of the notional system. Section 2.7.2 discusses the insertion of static fault-tolerance into the notional system. Section 2.7.3 details the use of AFT in this system as a logical progression of the insertion of static fault-tolerance.

2.7.1. AFT Application Architecture

The hardware and software architectures assumed in the notional example are detailed in Figures 2-1 and 2-2, respectively. The system detailed in these two figures represents a small portion of a fuller military BM/C³I system. However, even the small number of hardware and software nodes in this example will sufficiently show the effect AFT technology can have on these and similar systems.

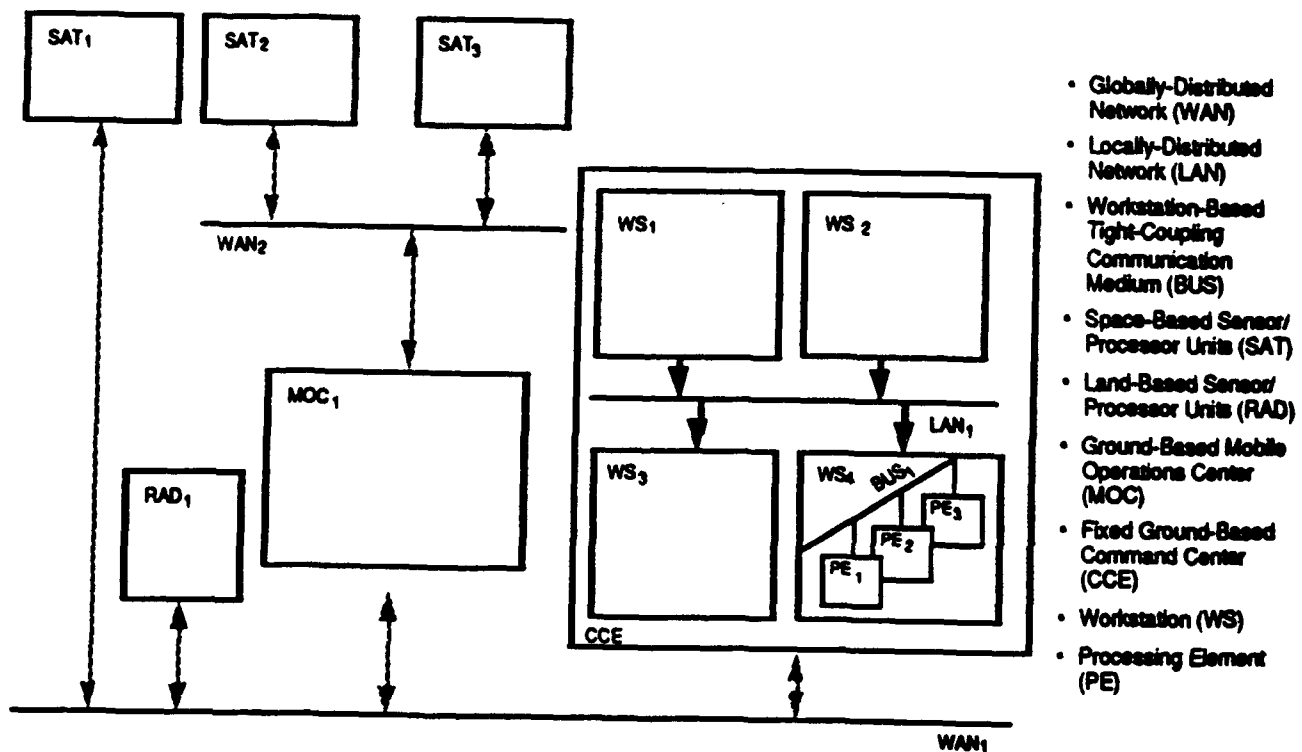


Figure 2-1
Notional Example Hardware Architecture

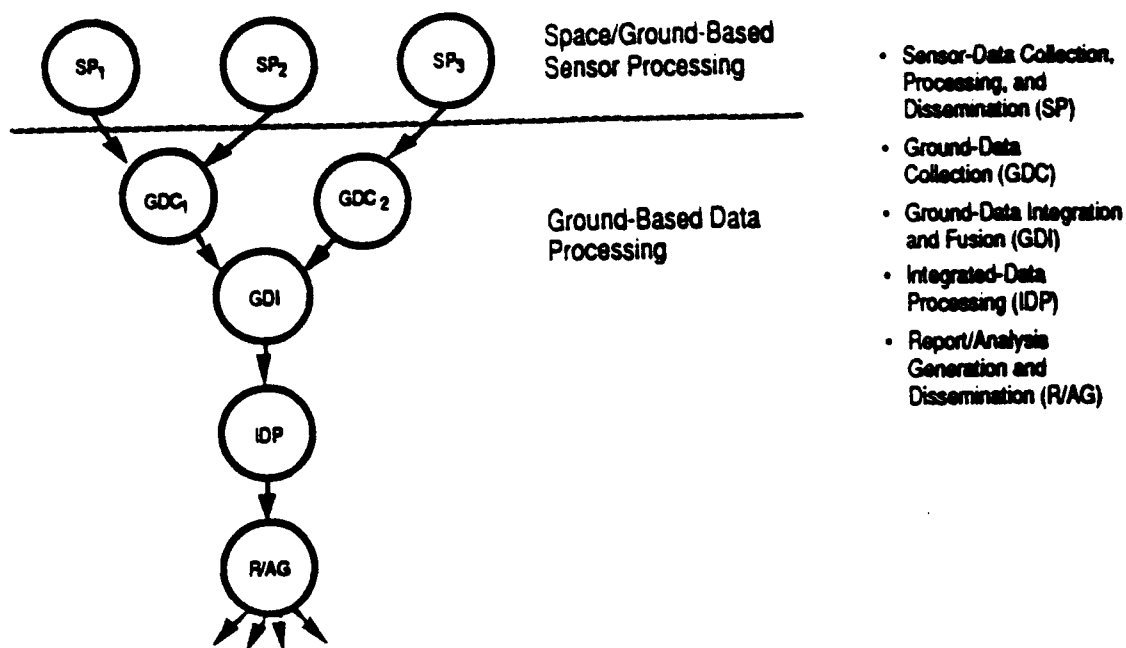


Figure 2-2
Notional Example Software Architecture

The hardware details a large, geographically-distributed system where individual elements communicate via Wide-Area Networks (WANs), Local-Area Networks (LANs), and multiprocessor interconnection networks. Processing hardware in this system consists of space-based sensor and processing units such as satellites (SAT), ground-based sensor and processing units such as radar (RAD), ground-based mobile processing units such as mobile operations centers (MOC), and ground-based fixed-processing units such as command centers (CCE). In finer detail, a processing element may be composed of some integration of single CPU workstation systems (WS) or multiprocessor workstations. Figure 2-1 shows one possible system composed of these elements.

The software for the notional system consists of both sensor processing and data processing components. Figure 2-2 shows a functional data flow of the software that will be run on this system. In this software a variety of separate sources perform sensor data collection and initial processing (SP). A fewer number of sources then perform data collection of the sensor output (GDC). Finally, a single source is responsible for ground-data integration and fusion of all sensor data (GDI). A single source is also responsible for integrated data post-processing on the fused sensor data (IDP). Finally, a single source is responsible for report and analysis generation and data dissemination (R/AG).

Figure 2-3 shows a mapping of the notional software architecture onto the notional hardware. The sensor processing (SP₁, SP₂, and SP₃) routines have been allocated to the sensor processors in the system (SAT₁, SAT₂, and RAD₁, respectively). Ground data collection (GDC) is performed by both a mobile operations center (MOC₁) and one workstation processor (WS₁) in the

command center (CCE). The remaining data processing routines (GDI, IDP, R/AG) have been allocated to workstations within the command center element (CCE).

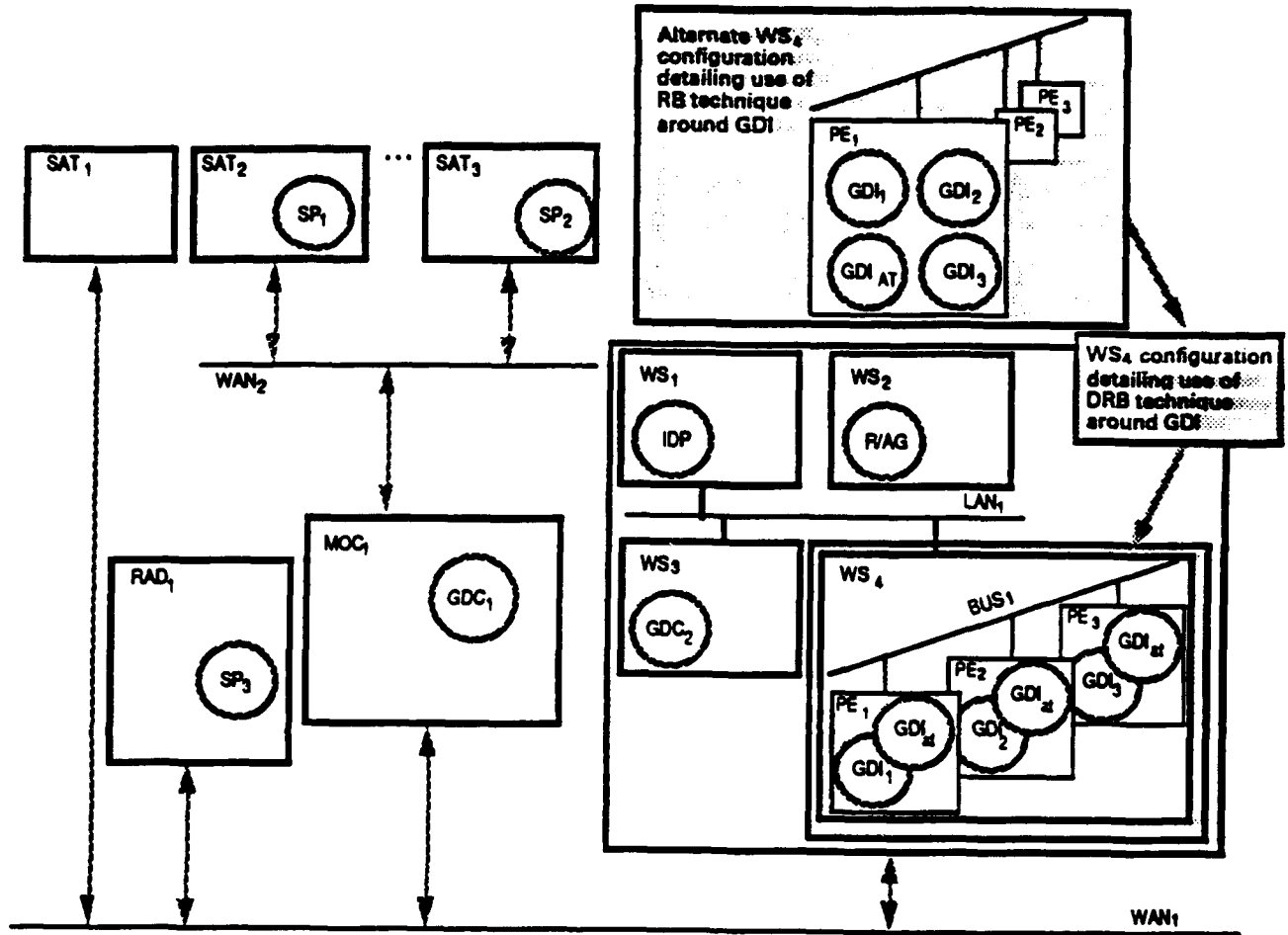


Figure 2.3
Notional Example Hardware/Software Mapping

Notice, the allocation of the ground data integration and fusion (GDI) function does not exactly resemble the single node structure shown in Figure 2-2. This is because Figure 2-2 only shows the simple data flow of the software, and does not allow for any fault tolerance to be expressed. The mapping of GDI onto WS4 in both alternate configurations will be explained in the next section.

It should be emphasized that the software architecture shown here is not necessarily the only software running on our notional hardware. The software shown consists of only those components that make up the logical data flow of one portion of a larger system. It is assumed that a number of additional auxiliary processing software elements are also running on the notional hardware. This issue will be of some importance as we continue to show how AFT can be used within this system.

2.7.2. Adding Static Fault Tolerance to the Notional Example

Before we can speak sensibly about the use of AFT in the notional example, we must first understand how and where the given system could be made fault tolerant in the traditional sense. For the sake of clarity and convenience, we will focus on one specific area where the use of AFT technology could benefit this system.

The focus will be on adding fault tolerance to the GDI function. Being one of the non-redundant nodes in a critical bottleneck of a C³I system, it is reasonable to expect a function such as GDI to be fault tolerant. In fact, one could easily argue that all functions in this notional example should have some level of fault tolerance. We simply focus on one example.

It is assumed that GDI is a fairly complex function that will meaningfully combine the incoming sensor data by some form of intelligent processing. Quite frequently multisensor data fusion problems have a number of different algorithms or software methods that can implement a solution. It should also be noted that multisensor fusion is not an exact science, and that different algorithms can be stronger in generating results for different portions of a total solution space. Some algorithms may even generate obviously incorrect answers for small portions of a solution space. In designing such a system, the costs and benefits of different solutions are evaluated and an "overall best" solution is picked.

Researchers have studied the use of techniques, such as recovery blocks (RB) and distributed recovery blocks (DRB), to provide fault tolerance against both common hardware faults as well as software faults as a result of improper algorithm implementation or algorithms that do not operate properly for an entire solution space. These techniques use a prioritized ordering of multiple designs and/or implementations of a solution and an acceptance test of the computed results.

The RB technique works on a single processor. The various solution algorithms are executed in priority order until the result of an algorithm passes the acceptance test. If an acceptance test fails, RB uses a backward error recovery strategy to run the next algorithm from its initial point. The RB technique is good at tolerating hardware faults resulting in erroneous or omitted computational results as well as design and implementation software faults.

Conversely, DRB uses a multiple processor configuration. All solution algorithms are executed simultaneously, and the results of the highest prioritized algorithm that passes the acceptance test are used. The DRB technique tolerates all the faults tolerated by RB in addition to hard crashes of either a hardware or software component. Also, the run-time of a DRB-based computation is only as long as the run-time of the longest executing single-solution implementation, plus some small overhead for acceptance testing and communication of results. This makes DRB computation desirable when predictable real-time performance is important.

Because of the similarity in these fault-tolerance techniques (FTTs), the term RB-class FTT is used to describe the use of either the RB or DRB technique. Given the assumptions about the type of processing being performed in GDI, and that GDI is a critical bottleneck through which all sensor data must pass, an RB-class FTT is used to provide the necessary fault tolerance in our notional example.

Figure 3-3 showed the notional hardware/software mapping using either the RB or DRB technique to implement a fault tolerant version of GDI, assuming that three different designs/implementations of the GDI solution were available. We label these three implementations GDI₁, GDI₂, and GDI₃. We also must assume that an acceptance test has been generated to properly determine if a solution is valid. We label the data flow node for the acceptance test GDI_{at}. In the RB implementation, GDI₁, GDI₂, GDI₃, and GDI_{at} were all mapped onto one of the processors, PE₁, available in WS₄. This leaves PE₂ and PE₃ available for auxiliary or background processing. Alternatively, in the DRB implementation, GDI₁, GDI₂ and GDI₃ are all mapped onto separate processors in WS₄. Local copies of GDI_{at} are also mapped onto each of the processors for efficiency. This would necessarily limit the amount of auxiliary processing done on PE₂ and PE₃.

2.7.3. Adding AFT to the Notional Example

The decision on whether to use the RB technique versus the DRB technique as the fault-tolerance mechanism for GDI provides an excellent example of the application of AFT. In command and control processing, the mechanisms that apply to a peacetime mode are not relevant to a battle mode. The change from a peacetime processing mode to a battle processing mode represents an external situation change in our dynamic operating environment. The external situation change from peacetime mode to battle mode subsequently prompts a change in system priorities.

In peacetime mode, it is important to maintain a high degree of consistency to verify any potential incoming hostile targets. In such a situation, soft deadlines are usually in effect because it could be acceptable to wait on or even lose an occasional data frame when the incoming target probability is extremely low. The soft deadline requirement then allows us to use processing cycles for miscellaneous auxiliary processing. The RB processing strategy fits this scenario perfectly. Only one processor is needed for GDI processing in this situation, leaving other processors free to handle auxiliary tasks.

If it is believed that the probability of incoming hostile targets is low, then we emphasize the capability to prevent false alarms that could have catastrophic results if incorrectly detected. This is one argument for the use of auxiliary processing, where the auxiliary processing could consist of various assurance checks, and hence, one additional recommendation for the use of an RB-class computation.

Conversely, in battle mode, time is at a premium, processing loads are extremely heavy, and consistency is of less concern than performance and functionality. During battle mode, the overall assumption is that incoming hostile targets are of a high probability and that high priority should be given to counteracting any target as soon as it is discovered.

With the change from peace mode to battle mode, system requirements have also changed. It is now most important to get maximum performance out of the system. Auxiliary processing is a lower priority than those functions that are absolutely essential to achieving system goals, which in this example are directed at tracking and counteracting an incoming hostile target.

The DRB processing strategy is more applicable to this situation because DRB provides assurances that the performance of the computations will be known, even in the occurrence of a fault. Because performance is highest priority in battle mode, auxiliary computation can be dropped if it will allow essential functions to be executed at maximum priority.

If AFT was used by such a system, both capabilities would exist. Furthermore, AFT, and the use of the AB Manager in particular, would allow the decision strategy and dynamic adjustment mechanisms to be isolated and properly modularized, maintaining good software design principles.

3. Demonstration

In the demonstration phase the abstract and theoretical concepts developed in the research portions of the AFT effort were applied to a prototype application. The resulting contract deliverable is referred to as the Interim Demonstration.

Throughout the AFT program, the Strategic Defense System (SDS) was used as a motivating application which drove the development of AFT requirements. In particular, the MONITOR function of the SDS Command Center Element (CCE) was the focal point. The MONITOR function is a central point through which all incoming raw telemetry data is collected, unified, operated on, and then disseminated. This makes MONITOR a single point of functional failure in the SDS system. Fault tolerance is, therefore, critical to MONITOR's operation.

A top level functional breakdown of MONITOR is shown in Figure 3-1. In fact, the information flow through this system follows a very common signal processing model of data collection, data processing, and report dissemination. The evolving SDS design currently identifies separate processors for each of three top level MONITOR subfunctions.



Figure 3-1
MONITOR Function Top Level Breakdown

The demonstration exhibited the viability of AFT concepts as applied to a BM/C3I signal processing stream modelled after the control structure of the SDS CCE MONITOR function. The system uses a simplified one-component AB Manager with knowledge of one potential adaptation.

Since Type 1 adaptations are generally considered the most difficult, it was decided that a Type 1 adaptation could best demonstrate the viability of AFT in a demonstration system which involved only one adaptation. Section 2.7 provides rational for the use of RB/DRB adaptations within a system like SDS. This rational was similarly applied to the PROCESS subfunction, and the Type 1 adaptation between RB and DRB was used in the demonstration.

Having chosen a suitable adaptation for demonstration, attention focused on the adaptation's causing scenario. That is, what stimulus will be used to effect demonstration system requirements to cause an adaptation to occur. Two potential scenarios for adaptation were evaluated.

The first scenario was analogous to the scenario detailed in the notional example of section 2.7 where adaptations were made in response to

reprioritized system requirements after incurring changes in the system's external environment. That is, raising emphasis on performance, and lowering emphasis on consistency as threat of attack increases.

In the second scenario, adaptations were based on an evaluation of the time spent in backward error recovery (with RB) as the error rate of the system increased. As error rate increases so does the time spent in error recovery – possibly forcing deadlines to be missed with greater frequency. Thus, the rationale for using DRB over RB increases too.

As a result of discussions with Rome Laboratory it was decided to use the second scenario in the AFT demonstration.

Section 3.1 describes the architecture of the demonstration system as installed at Rome Laboratory's DISE testbed facility. Section 3.2 details the critical components used in development of the Interim Demonstration. Section 3.3 describes overall result.

3.1. Demonstration Architecture

There are two ways to describe the architecture of the AFT demonstration system: the virtual level and the physical level. At the virtual level, the demonstration simulates a BM/C³I application running on a distributed system of 5 nodes. At the physical level, this simulation consists of nine separate processes running across three physical processors.

The architecture of the virtual BM/C³I application models the functional structure shown in structure shown in Figure 3-1. However, this model needed to be supplemented with additional top-level functionality for insertion of AFT technology.

Figure 3-2 is a screen dump of one of the windows of the interim demonstration. This figure shows the high-level breakdown of the BM/C³I application at the virtual level. At the hardware level, the virtual system contains 5 processors labeled *(Virtual) Hardware Node 1* through *(Virtual) Hardware Node 5* in the figure. At the software level, the virtual system still models the COLLECT, PROCESS, and DISSEMINATE functions of the CCE MONITOR. A second level breakdown of PROCESS is required to show the multiple try-blocks that are required in RB-class fault tolerance techniques.

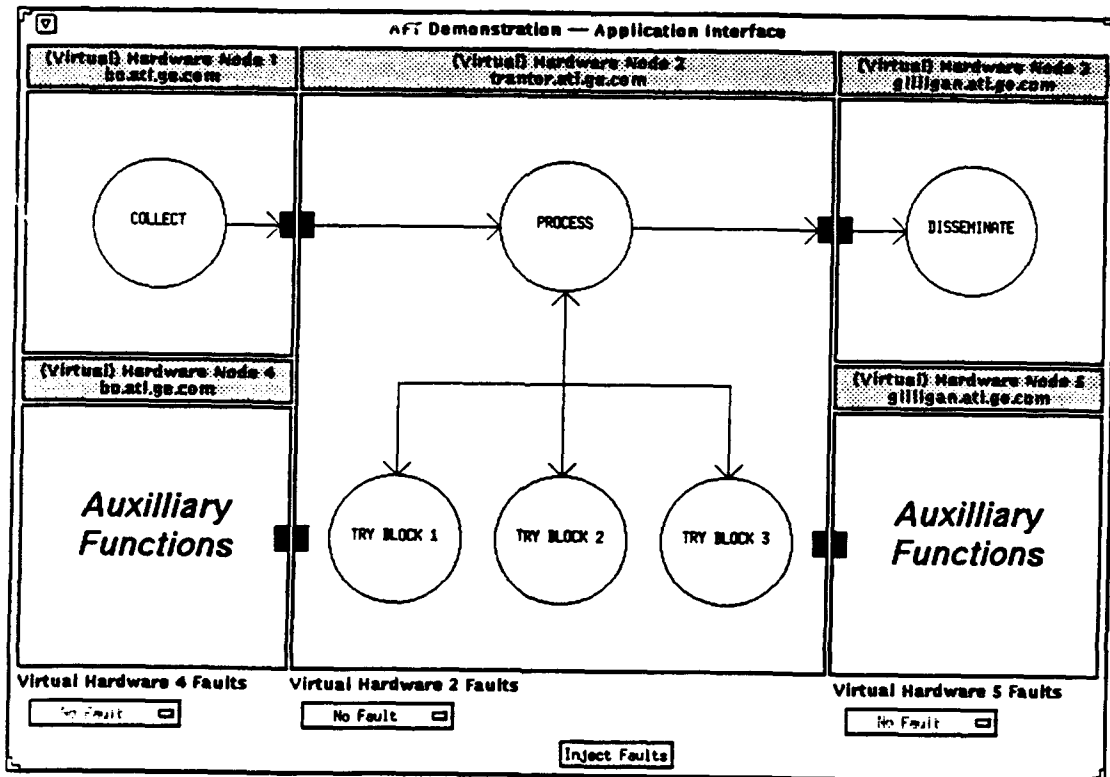


Figure 3-2
High-Level Structure of AFT Interim Demonstration
Virtual Application View (RB mode)

Note that when the system is running in RB mode, all three try-blocks exist on the same processor as the parent function, PROCESS. This is the scenario shown in Figure 3-2. During this mode of operation, (Virtual) Hardware Node 4 and (Virtual) Hardware Node 5 are considered to be running useful but expendable auxiliary functions. However, Figure 3-3 shows the high-level breakdown of the same window when an adaptation has caused the application to switch to DRB mode, replacing the auxiliary functionality with one of the try-blocks.

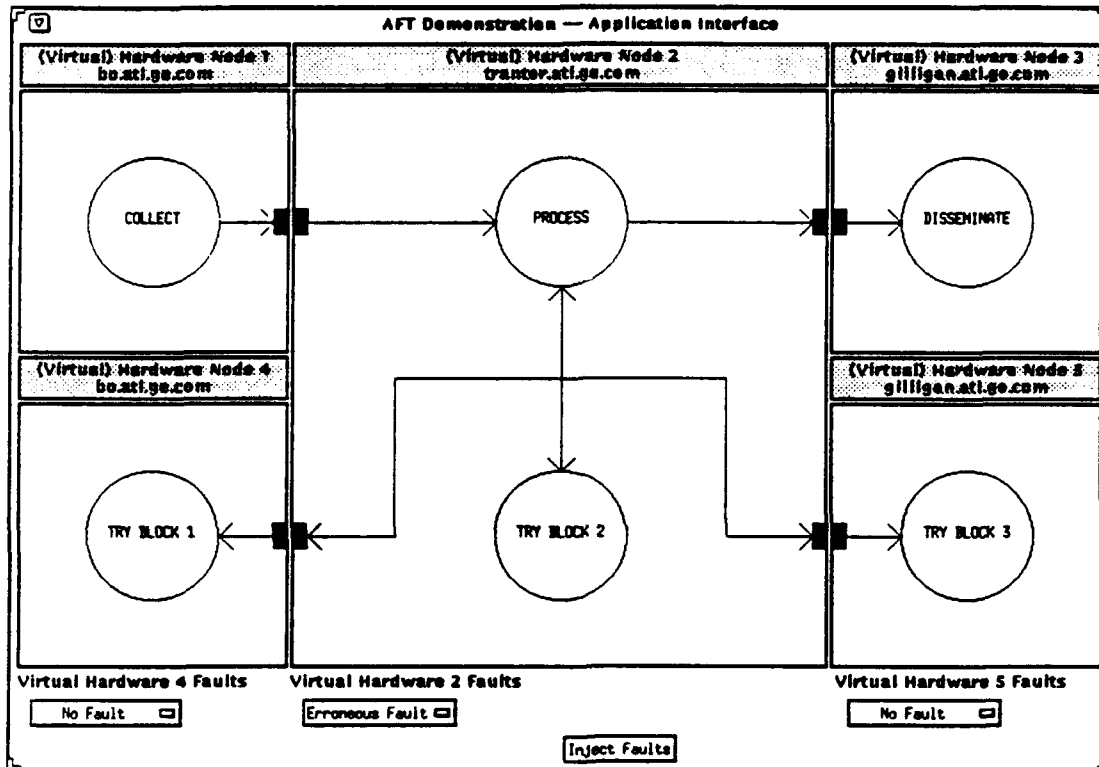


Figure 3-3
High-Level Structure of AFT Interim Demonstration
Virtual Application View (DRB mode)

The architecture of the demonstration's physical level is a bit more detailed. Figure 3-4 shows the high-level physical structure of the AFT demonstration.

The physical system consists of only three hardware nodes. At the Rome Laboratory installation, these nodes were Orion, Rigel, and Janus. To properly emulate the distributed processing of the 5 node virtual model on a 3 node physical system (Virtual) Hardware Node 2, (Virtual) Hardware Node 4, and (Virtual) Hardware Node 5, were required to be on separate physical processors. This was to achieve true physically distributed processing of all try-blocks when a system was in DRB mode. The simulation of (Virtual) Hardware Node 1 and (Virtual) Hardware Node 3 could exist on any of the three physical processors.

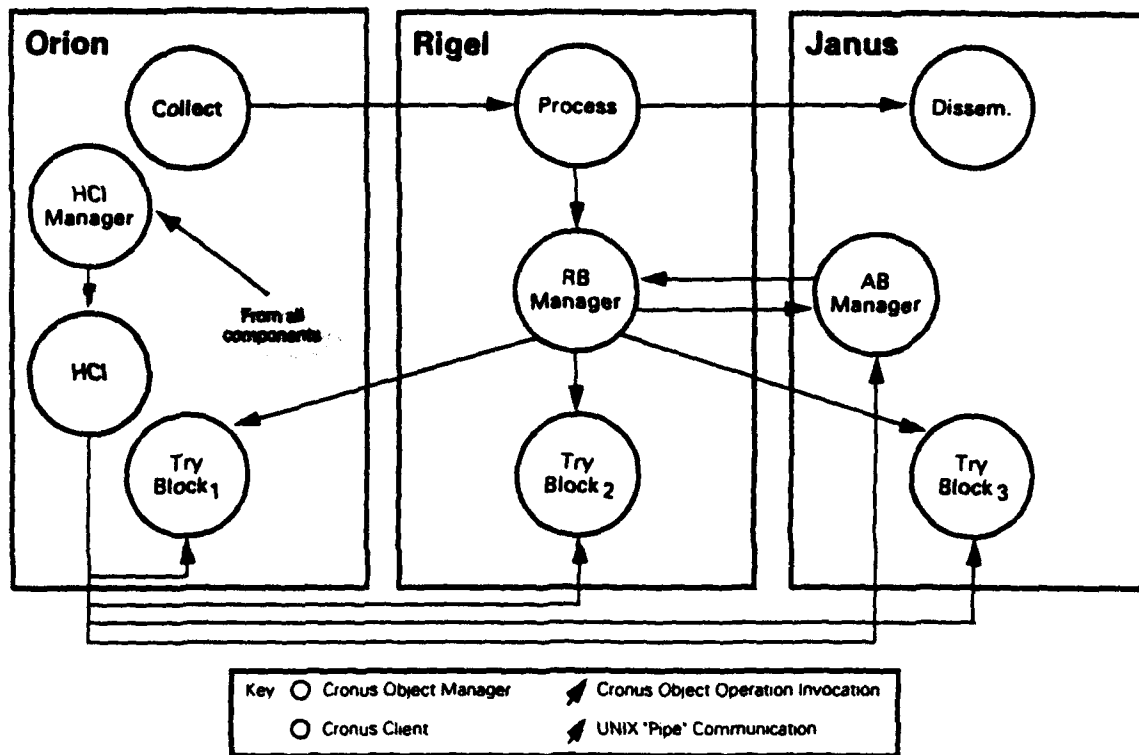


Figure 3-4
High-Level Structure of AFT Interim Demonstration
Physical View

There are nine separate software processes comprising the physical demonstration system. Separate software processes still exist representing the COLLECT, PROCESS, and DISSEMINATE functions.

An AB Manager process also exists (as would be required in any AFT system). This process continues the adaptive behavior decision making mechanisms for the demonstration.

The RB Manager module is a process which implements the black-box interface to RB-class fault tolerance techniques.

The three modules labelled "Try Block 1" through "Try Block 3" represent try-block server processes which exist at each physical node. Upon command, a try block server can execute any of the requested try-blocks on input data. Hence, the capability exists to execute any try-block at any given node. Thus, in RB mode, only Try Block Server 2 executes try-blocks. If execution of the first try-block fails, Try Block Server 2 executes the second try block, followed by the third if necessary. However, in DRB mode, Try Block Server 1 would execute the first try-block, simultaneously, Try Block Server 2 would execute the second try-block, and Try Block Server 3 would execute the third try-block. The additional capability to execute all try-blocks at all try-block servers is unused at this time, but should prove useful for future research in this field.

Lastly, the demonstration's human-computer interface is represented by two modules labelled HCI and HCI Manager. Although the use of the Cronus Distributed System toolset and X-Windows will be explained in Section 3.2, the rationale for dividing the human-computer interface functionality is related to their roles in the demonstration. The HCI module is implemented in X-Windows and contains the portions of the human-computer interface which deal with workstation, mouse, and keyboard I/O (for example, the drawing of output graphics). The HCI Manager module is implemented in Cronus, and is the communication interface between the all other portions of the demonstration and the HCI module. This is necessary because both Cronus and X-Windows want to take entire control of an application process. Thus, the demonstration's HCI functionality was partitioned into separate Cronus (HCI Manager) and X-Windows (HCI) processes which communicate via a common Unix pipe interface.

3.2. Components Used in Demonstration Development

The interim demonstration was developed entirely on the Sun-4 platform at GE ATL. Ports to the Sun-3 platform should only require recompilation. As stated in Section 3.1, the system required 3 separate processors to run, however, up to 7 processors could be used. A simple database of physical software modules to physical hardware nodes needs to be constructed prior to demonstration execution. This database allows reconfiguration of software modules to hardware nodes without recompilation.

The Cronus distributed system development environment was used to formalized the distributed element communication model. It is also believed that the Cronus tasking model will be highly useful for future AFT development. Most of the software modules were implemented as Cronus managers.

The X-Windows system was used for human-computer interface development. In particular, the Motif interface standard was adhered to for the greatest degree of future portability.

Also, to ease the turn-around time in demonstration development, the Widget Creation Library (WCL) toolset was used in human-computer interface development. WCL provides a greater degree of flexibility in the development of X-Windows applications by enabling much of the structure of the resulting interface to be specified in a resource file, rather than directly into the code. Hence, major changes to the structure of an interface developed with X-Windows can be made without lengthy recompilations.

3.3. Demonstration Results

The AFT interim demonstration is an interactive platform for the study of AFT concepts and their related performance. The interim demonstration enables users to interactively inject faults (either crash, erroneous, or omission faults) into the simulated BM/C³I application, and observe the effects of adaptive behavior management on the organization of try-blocks which compute the PROCESS function.

During the demonstration, frames of data pass from COLLECT, to PROCESS, and then eventually onto DISSEMINATE. The real-time goal is to have each frame take no longer than one second to pass from COLLECT through to DISSEMINATE. If this real-time limit is violated, then a *frame error* is signalled. A frame error is also signalled if the real-time goal is achieved, but the final result sent to DISSEMINATE is incorrect, as determined by the acceptance tests. This means that all try-blocks have failed, but have done so within the real-time limit.

Note that a *try-block error* can occur without a frame error occurring, as long as at least one try-block is successful, and a correct result is reached within the real-time limit.

The AB Manager uses a history-based heuristic to determine if adaptation is needed. A running window of history is kept across the number of frame errors which have occurred, and the average time spent in computation of the application stream. Once the number of frame errors within the history window and the average computation time go beyond given threshold values, an adaptation is made from RB to DRB. Both number of frame errors and average computation time are used to avoid random spikes in either of the two data sets. Similar heuristics are used for the DRB to RB adaptation, although the threshold values differ to avoid thrashing across a state represented by a single threshold point.

The AFT interim demonstration does prove the viability of AFT concepts within the chosen BM/C³I application. Interestingly, the use of similar heuristics with different threshold values did not prevent thrashing, but only slowed down the process. Future efforts will study the use of multiple adaptations, adaptations of Type 2 through Type 5, and variations within both optimistic and pessimistic adaptation strategies to prevent thrashing.

DISTRIBUTION LIST

addresses	number of copies
MR. THOMAS LAWRENCE RL/C3AB BLDG #3 525 BROOKS ROAD GRIFFISS AFB NY 13441-4505	20
MR. LARRY ALEXANDER MARTIN MARIETTA ADVANCED TECHNOLOGY LAB/BLDG 145 MOORESTOWN CORPORATE CENTER MOORESTOWN NJ 08057	5
RL/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY GRIFFISS AFB NY 13441-4514	1
ADMINISTRATOR DEFENSE TECHNICAL INFO CENTER DTIC-FDAC CAMERON STATION BUILDING 5 ALEXANDRIA VA 22304-6145	2
BALLISTIC MISSILE DEFENSE ORGANIZATION 7100 DEFENSE PENTAGON WASH DC 20301-7100	2
RL/C3AB 525 BROOKS RD GRIFFISS AFB NY 13441-4505	1
NAVAL WARFARE ASSESSMENT CENTER GIDEP OPERATIONS CENTER/CODE JA-50 ATTN: E RICHARDS CORONA CA 91718-5000	1
HQ ACC/DRIY ATTN: MAJ. DIVINE LANGLEY AFB VA 23665-5575	1

ASC/ENEMS 1
WRIGHT-PATTERSON AFB OH 45433-6503

WRIGHT LABORATORY/AAAI-4 1
WRIGHT-PATTERSON AFB OH 45433-6543

WRIGHT LABORATORY/AAAI-2 1
ATTN: MR FRANKLIN HUTSON
WRIGHT-PATTERSON AFB OH 45433-6543

AFIT/LDEE 1
BUILDING 642, AREA B
WRIGHT-PATTERSON AFB OH 45433-6583

WRIGHT LABORATORY/MTEL 1
WRIGHT-PATTERSON AFB OH 45433

AAMRL/HE 1
WRIGHT-PATTERSON AFB OH 45433-6573

AIR FORCE HUMAN RESOURCES LAB 1
TECHNICAL DOCUMENTS CENTER
AFHRL/LRS-TDC
WRIGHT-PATTERSON AFB OH 45433

AUL/LSE 1
BLDG 1405
MAXWELL AFB AL 36112-5564

US ARMY STRATEGIC DEF 1
CSSD-IM-PA
PO BOX 1500
HUNTSVILLE AL 35807-3801

COMMANDING OFFICER 1
NAVAL AVIONICS CENTER
LIBRARY D/765
INDIANAPOLIS IN 46219-2189

Commanding Officer 1
NCCOSC RDTE Division
Code 0274B, Tech Library
53560 Hull Street
San Diego CA 92152-5001

CMDR 1
NAVAL WEAPONS CENTER
TECHNICAL LIBRARY/C3431
CHINA LAKE CA 93555-6001

SPACE & NAVAL WARFARE SYSTEMS COMM 1
WASHINGTON DC 20363-5100

CDR, U.S. ARMY MISSILE COMMAND 2
REDSTONE SCIENTIFIC INFO CENTER
AMSMI-RD-CS-R/ILL DOCUMENTS
REDSTONE ARSENAL AL 35898-5241

ADVISORY GROUP ON ELECTRON DEVICES 2
ATTN: DOCUMENTS
2011 CRYSTAL DRIVE, SUITE 307
ARLINGTON VA 22202

LOS ALAMOS NATIONAL LABORATORY 1
REPORT LIBRARY
MS 5000
LOS ALAMOS NM 87544

AEDC LIBRARY 1
TECH FILES/MS-100
ARNOLD AFB TN 37389

COMMANDER/USAISC 1
ATTN: ASOP-DO-TL
BLDG 61801
FT HUACHUCA AZ 85613-5000

AIR WEATHER SERVICE TECHNICAL LIB 1
FL 4414
SCOTT AFB IL 62225-5458

AFIWC/MSO 1
102 HALL BLVD STE 315
SAN ANTONIO TX 78243-7016

SOFTWARE ENGINEERING INST (SEI) 1
TECHNICAL LIBRARY
5000 FORBES AVE
PITTSBURGH PA 15213

DIRECTOR NSA/CSS 1
W157
9800 SAVAGE ROAD
FORT MEADE MD 21055-6000

NSA 1
E323/MC
SAB2 DOOR 22
FORT MEADE MD 21055-6000

NSA 1
ATTN: D. ALLEY
DIV X911
9800 SAVAGE ROAD
FT MEADE MD 20755-6000

DOD 1
R31
9800 SAVAGE ROAD
FT. MEADE MD 20755-6000

DIRNSA 1
R509
9800 SAVAGE ROAD
FT MEADE MD 20775

DIRECTOR 1
NSA/CSS
R08/R & E BLDG
FORT GEORGE G. MEADE MD 20755-6000

DOD COMPUTER CENTER 1
C/TIC
9800 SAVAGE ROAD
FORT GEORGE G. MEADE MD 20755-6000

ESC/IC 1
50 GRIFFISS STREET
HANSCOM AFB MA 01731-1619

ESC/AV 1
20 SCHILLING CIRCLE
HANSCOM AFB MA 01731-2816

DCMAO/GWE 1
ATTN: JOHN CHENG
US COURTHOUSE/SUITE B-34
401 N MARKET
WICHITA KS 67202-2095

FL 2807/RESEARCH LIBRARY 1
CL AA/SULL
HANSCOM AFB MA 01731-5000

TECHNICAL REPORTS CENTER 1
MAIL DROP D130
BURLINGTON ROAD
BEDFORD MA 01731

DEFENSE TECHNOLOGY SEC ADMIN (DTSA) 1
ATTN: STTD/PATRICK SULLIVAN
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

ADVANCED SYSTEM TECHNOLOGIES 1
ATTN: DUANE R. BALL
5113 LEESBURG PIKE, SUITE 514
FALLS CHURCH VA 22041

ODYSSEY RESEARCH ASSOCIATES, INC. 1
ATTN: DOUG WEBER
301A HARRIS B. DATES DR
ITHACA NY 14850-1313

SRI INTERNATIONAL
333 RAVENSWOOD AVE
MENLO PARK CA 94025

1

CONCURRENT COMPUTER CORP
ATTN: RAYMOND CLARK
1 TECHNOLOGY WAY
WESTFORD MA 01886

1

U. S. ARMY CECOM
ATTN: LAKSHMI V. REBBAPRAGADA
CENTER FOR C3 SYSTEMS
AMSEL-RD-C3-IR
FT MONMOUTH NJ 07703

1

NRAD
ATTN: LES ANDERSON
271 CATALINA BLVD, CODE 413
SAN DIEGO CA 92151

1

DARPA/ISTO
ATTN: BRIAN BOESH
1400 WILSON BLVD
ARLINGTON VA 22209-2308

1

AFSCS/SRER SUITE 139
ATTN: GLENN ARMSTRONG
250 HALL BLVD
SAN ANTONIO TX 78243-7063

1

NRA/R232
ATTN: DANIEL W. ATKINSON
9800 SAVAGE RD
FT MEADE MD 20755-6000

1

TRUSTED INFORMATION SYSTEMS, INC.
ATTN: WILLIAM C. BARKER
3060 WASHINGTON RD
GLENWOOD MD 21738

1

BBN SYSTEMS AND TECHNOLOGIES CORP
ATTN: JAMES C. BERETS
10 MOULTON STREET
CAMBRIDGE MA 02138

1

DIRNSA 1
ATTN: LEN BINNS
R232
9800 SAVAGE RD
FT. MEADE MD 20755

HONEYWELL, INC. 1
ATTN: MIKE BOSQUEZ
3660 TECHNOLOGY DRIVE
MINNNEAPOLIS MN 55418

AFSCS/SRER 1
ATTN: JOHN BRES
KELLY AFB TX 78219

NSA/V5 1
ATTN: DR. JOHN CAMPBELL
9800 SAVAGE RD
FT. GEORGE G. MADE MD 20755-6000

SYRACUSE UNIVERSITY 1
ATTN: SHIU-KAI CHIN
SYRACUSE NY 13244-4100

DEFESNE INFORMATION SYSTEMS AGENCY 1
ATTN: LT COL RICHARD HEPWORTH
MLS PROGRAM OFFICE
3701 N. FAIRFAX DRIVE
ARLINGTON VA 22209

ATTN: PATRICA BASKINGER 1
TASC
555 FRENCH ROAD
NEW HARTFORD NY 13413-0895

ATTN: MIKE DAVIS 1
SRI INTERNATIONAL
333 RAVENSWOOD AVE
MENLO PARK CA 94025-3493

SRI INTERNATIONAL 1
ATTN: JACK GOLDBERG
333 RAVENSWOOD AVE
MENLO PARK CA 94025-3423

MITRE CORPORATION 1
ATTN: HARRIET GOLDMAN
BURLINGTON RD
BEDFORD MA 01730

SECURE COMPUTING TECHNOLOGY CORP 1
ATTN: DR. J. THOMAS HAIGH
1210 WEST COUNTY RD E, SUITE 100
ARDEN HILLS MN 55112

ORA CORPORATION 1
ATTN: BRET HARTMAN
301A HARRIS B. DATES DR.
ITHACA NY 14850-1313

NAVAL SYSTEMS WEAPONS CENTER 1
ATTN: STEVE HOWELL/U33
10901 NEW HAMPSHIRE AVE
SILVER SPRINGS MD 20903-5000

AFCSC/SRER 1
ATTN: DA HURER
SAN ANTONIO TX 78234-5000

SDIO/SDA 1
ATTN: RICHARD IIEF
THE PENTAGON
WASH DC 20301

GEORGE MASON UNIVERSITY 1
ATTN: SUSHIL JAJODIA
ISSE DEPT
FAIRFAX VA 22030-4444

NSA/R206 1
ATTN: LT COL JOE JAREMKO
FT MEADE MD 20755-6000

NRAD/E 413 1
ATTN: RUSSELL JONSTON
271 CATALINA BLVD
SAN DIEGO CA 92152-5000

AFSC/XTK
ATTN: CAPT GIL LEE
ANDREWS AFB MD 20334

1

US ARMY CECOM
ATTN: JENNY LEE
AMSEL-ORD-C3-CC-A
FT MEADE NJ 07703

1

APPLIED RESEARCH & ENGINEERING
ATTN: DONALD M. LESKIW
435 ARBORETUM WAY
BURLINGTON MA 01803

1

SRI INTERNATIONAL
ATTN: TERESA LUNT
COMPUTER SCIENCE LABORATORY
333 RAVENSWOOD AVE
MENLO PARK CA 94025-3493

1

KNOWLEDGE BASE SYSTEMS LAB
ATTN: DR. RICHARD MAYER
DEPT INDUSTRIAL ENGINEERING
TEXAS A&M
COLLEGE STATE TX 77843

1

MITRE CORP
ATTN: CATHERINE MCCOLLUM
7525 COLSHIRE DRIVE
MCLEAN VA 22101-3481

1

SECURE COMPUTING CORPORATION
ATTN: CORNELIA MURPHY
1210 WEST COUNTY RD E
SUITE 100
ARDEN HILLS MN 55112

1

MITRE CORP
ATTN: LOUANNA NOTARGIACOMO
7525 COLSHIRE DR
MCLEAN VA 22102-3481

1

UNISYS CORPORATION
ATTN: HANS W. POLZER
12010 SUNRISE VALLEY DR
RESTON VA 22091

1

HONEYWELL (MS 2350)
ATTN: SATYA PPARHAKER
3660 TECHNOLOGY DRIVE
MINNEAPOLIS MN 55418

1

US ARMY CECOM
ATTN: JOHN PREUSSE
AMSEL-RD-C3-IS-P
FT MONMOUTH NJ 07703

1

MITRE CORP
ATTN: MYRA JEAN PRELLE
BURLINGTON RD
BEDFORD MA 01730

1

US ARMY CECOM
ATTN: JOHN RUSHMEYER
AMSEL-RD-C3-CC-A
FT MONMOUTH NJ 07703

1

UNIVERSITY OF MARYLAND
ATTN: KEY SALEM
INST. FOR ADVANCED COMPUTER STUDIES
DEPT OF COMPUTER SCIENCE
COLLEGE PARK MD 20742

1

DIR, NSA/R23
ATTN: O.SAMI SAYDJARI
9800 SAVAGE RD
FT MEADE MD 20755-6000

1

SRI INTERNATIONAL
ATTN: LOUS C. SCHREIER
333 RAVENSWOOD AVE
MENLO PARK CA 94025-3493

1

TRUSTED INFORMATION SYSTEMS, INC.
ATTN: JOHN SEBES
444 CASTRO STREET, SUITE 800
MOUNTAIN VIEW CA 94041

1

ORA CORPORATION
ATTN: MAUREENN STILLMAN
301A HARRIS B. DATES DRIVE
ITHACA NY 14850-1313

1

GE AEROSPACE
ATTN: MIKE SUTTEN
RT 38 BLDG 145
MOORESTOWN CORPORATE CENTER
MOORESTOWN NJ 08057

1

INFOSYSTEMS TECHNOLOGY
ATTN: DR. CHARLES TESTA
6303 IVY LANE
GREENBELT MD 20770

1

UNIVERSITY OF MINNESOTA
ATTN: ANAND TRIPATHI
DEPT OF COMPUTER SCIENCE
MINNEAPOLIS MN 55455

1

TRUSTED INFORMATION SYSTEMS INC.
ATTN: STEPHEN T. WALKER
3060 WASH RD (RT 97)
GLENWOOD MD 21738

1

DIRNSA
ATTN: MICHAEL R. WARE
DOD, NSA/CSS (R23)
FT. GEORGE G. MEADE MD 20755-6000

1

***MISSION
OF
ROME LABORATORY***

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

- a. Conducts vigorous research, development and test programs in all applicable technologies;
- b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;
- c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;
- d. Promotes transfer of technology to the private sector;
- e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.