Workshop on Object-Oriented Reliable Distributed Systems, Newport Beach, CA, February, 1997

An Approach for Adaptive Fault-Tolerance in Object-Oriented Open Distributed Systems

Eltefaat Shokri Herbert Hecht Patrick Crane
SoHaR Incorporated, Beverly Hills, CA

Jerry Dussault
USAF Rome Laboratory

K. H. (Kane) Kim
University of California, Irvine

Abstract

Effective fault-handling in emerging complex distributed applications requires the ability to dynamically adapt resource allocation and fault-tolerance policies in response to possible changes in environment, application requirements, and available resources. This paper reports an effort on design and implementation of an adaptive fault-tolerance middleware (AFTM) using a CORBA-compliant object request broker resting on the Solaris open system platform. The paper also briefly discusses the essential capabilities of AFTM, the overall system architecture, and its design decisions.

1. Introduction

The new generation of complex mission-critical applications (such as military planning and monitoring applications) are inherently distributed and must operate in highly dynamic environments. In addition, timing and dependability requirements of these applications may vary during different phases of their life-span. In such large-scale complex systems, fault-handling and resource management capabilities cannot be statically configured mainly because the amount of resources required to meet the application needs with statically configured fault-handling functions becomes prohibitively large. In other words, fault-handling and resource management functions in these systems must dynamically adapt themselves in response to the changes in the environment, the application demands, and the system resources. Therefore, the technology called Adaptive Fault-Tolerance [1] (AFT) is an essential feature for emerging high complexity mission-critical systems.

There have been several research and development efforts on adaptive fault-tolerance carried out in academia, as well as in government laboratories [2,3,4]. Main contribution of these efforts was on identifying conceptual frameworks and potential applications of the AFT concept. However, efficient implementation models for both local-area and wide-area distributed real-time systems remains to be explored. Moreover, most of those demonstration or prototype systems were implemented in non-standard software/hardware platforms, making porting of the systems to other hardware/software environments a difficult task. There remains a need for developing an AFT management subsystem which possesses the following important characteristics [5]:

(i) The AFT subsystem should be implemented in open standard software/hardware platforms,

(ii) It should act as an interface between the application and the underlying software layers (i.e., acting as a middleware), *transparently* monitor the application behaviors as well as the availability of resources, and *adaptively* reconfigure the system resources accordingly,

(iii) It should include a *generic* adaptation policy suitable for a wide range of distributed applications.

Considering the degree of complexity of developing such a middleware, the state of the art in software engineering methodologies should be employed. Recent software engineering experience has demonstrated that

object-oriented design and programming techniques [6], if used with care, possess the potential for reducing software complexity and improving dependability and maintainability of complex software systems. Recently, the Object Management Group (OMG) created a standard specification for developing distributed object-oriented applications. The standard was named the Common Object Request Broker Architecture (CORBA) [7]. CORBA provides a standard specification for location-transparent object interactions among distributed client-server objects. A CORBA-compliant communication middleware is a promising platform for implementing distributed applications, since it provides simple interfaces for interactions among objects residing in different computing nodes.

The need to investigate, design, and develop distributed systems with the AFT capabilities for distributed real-time applications has motivated the USAF Rome Laboratory to sponsor a research and development project for building an AFT management (AFTM) middleware on top of a CORBA-compliant distributed object support middleware. This paper briefly presents an abstract architecture of such a fault-tolerance middleware under development.

The paper starts in Section 2 with a discussion on the desirable attributes of an AFTM middleware. Section 3 discusses various components of the AFTM architecture being implemented. Section 4 discusses the implementation issues of the middleware. The fault-tolerant execution modes supported by the AFTM architecture are discussed in Section 5. Section 6 is a conclusion.

2. Desirable Attributes of an Adaptive FT Middleware

Although the current CORBA specification does not provide a sufficient set of capabilities for supporting real-time applications, a direction for realizing an effective real-time CORBA has become clear. We believe that such an extension should enable the designer to design the application as a set of real-time objects [10], i.e., objects which have time-triggered methods as well as service methods triggered by calls from client objects.

2.1 Essential Core Attributes of the AFTM Middleware

A suitable AFTM middleware should enhance system reliability, performance, survivability, and effectiveness through the following essential capabilities:

(i) **Provide significantly improved resource utilization:** An AFT approach is viable only if it can significantly improve utilization of available resources compared to conventional fault-tolerance mechanisms, while providing an acceptable degree of reliability for critical components. Moreover, the overhead incurred by the adaptive subsystem should be kept minimal.

(ii) **Provide fault-tolerance for all non-negligible fault types:** An important step in designing a fault-tolerance system is to identify failures which have non-negligible occurrence probability. One difficult decision here is whether to include software failures in the set of credible failure modes. Since experience has shown that the possibility of software design faults cannot be ignored in large-scale applications [9], AFTM will provide a variety of fault-handling provisions for tolerating both hardware and software faults. However, the application designer will inform AFTM whether software faults should be considered by AFTM or not.

(iii) **Respond to changes in the environment, system, and user profile:** Changes in environment may be manifested in various forms. For example, the degree of environmental hostility may increase during various phases of the application life (e.g., the probability of parts of a fighter being damaged is greatly increased during an attack), leading to a higher probability of physical failures of hardware components. Another example is that a spacecraft mission profile may include time-periods when long response times are acceptable, while there are other phases in which responses must be quite rapid (e.g., during a planet flyby). Changes in the internal state of the system are mainly due to failures in either the software or hardware components of a system and affect the availability of various system resources. The user/application demands may also vary during different phases of the application. The adaptation mechanism must effectively respond to these possible changes within an acceptable time-period.

(iv) **Consider application-specific adaptation parameters in the decisions**: The user and/or the developer of an application may provide application-specific parameters which affect the adaptation decisions. An effective AFT manager should be able to accept these parameters during the execution of the application and consider them in the decision making process.

(v) **Facilitate various adaptation modes:** A suitable adaptation policy should consider various modes of

adapting the system to current system states. The adaptation can be envisioned, for example, in one of the following ways: (1) switching from one fault-handling mode to another, (2) modifying parameters of the currently-used fault-handling mode, and (3) modifying service attributes.

(vi) Provide required services in an application-transparent fashion: The AFTM middleware should be implemented as a separate layer and must have a simple and clear interface with the application. The application should then interact with the AFTM layer only through these interfaces.

Figure 1 illustrates a candidate set of input parameters provided by the user and/or the run-time systems to facilitate realization of the above capabilities.

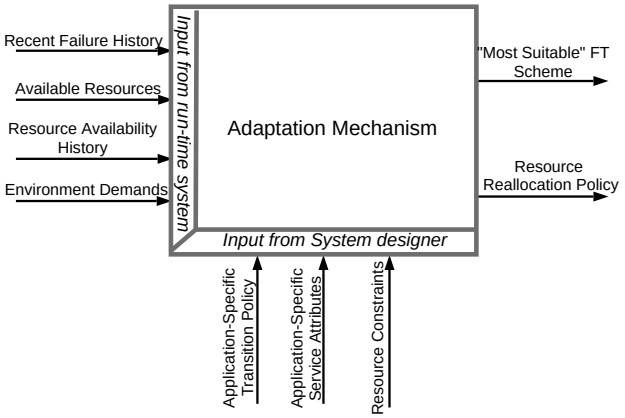

Figure 1. A candidate set of inputs to the adaptation mechanism

2.2 Secondary Attributes of the AFTM Middleware

In order to make AFTM more suitable for large-scale widely-distributed real-time applications the adaptation manager should be able to interact with the user and modify its policy according to the user commands. To facilitate this interaction with the user, the AFTM middleware should possess the following capabilities [5,8]:

(i) Automatic versus manual adaptation control: It is desirable for the AFTM middleware to allow the user to modify or override automatic adaptation decisions. This gives the user an extra capability to govern the resource allocation when the system enters an unexpected anomalous state.

(ii) Provide a system status information to the user: To effectively interact with the AFTM middleware, the user should be provided with information regarding the current state of a selective subset of system resources as well as recent events. The AFTM middleware should provide a graphical monitoring subsystem which allows the user to probe into the system and request specific information about the application as well as various AFTM components.

(iii) LAN to WAN scalability: In order to be applicable to a wide variety of applications, the AFTM middleware will be scalable and easily adapted to both local-area and wide-area networks.

3. Architecture of the AFTM Middleware

The overall architecture of the AFTM middleware with the capabilities discussed in Section 2 is depicted in Figure 2.

The first prototype implementation of the AFTM middleware will run in a network of Solaris workstations which adheres to industry standards such as the Posix 1003.1b thread library. We have chosen Solaris over the more mature SunOS because (i) it provides an easy-to-use application-level concurrency (multi-threaded programming), and (ii) "soft" real-time performance can be achieved using Solaris real-time thread facilities.

As shown in Figure 2, AFTM rests on a CORBA-compliant object request broker (ORB). Over the past several years, CORBA distributed object standards have been adopted by a rapidly growing number of users. Its growing popularity stems from the fact that (i) it makes development of distributed application easier by providing high-level and simple inter-object communication services, and (ii) it facilitates the interoperability among various distributed object tools.

To provide real-time facilities needed by the AFTM components as well as the application, we developed a layer (on top of a CORBA-compliant Object Request Broker) for supporting RTO.k real-time objects [10]. An RTO.k is an extension of conventional objects with the following additional features:

a) Each RTO.k may have methods which are activated spontaneously upon arrival of certain time-points (time-triggered methods), in addition to the conventional service methods activated by client objects.

b) Each method is associated with a completion deadline.

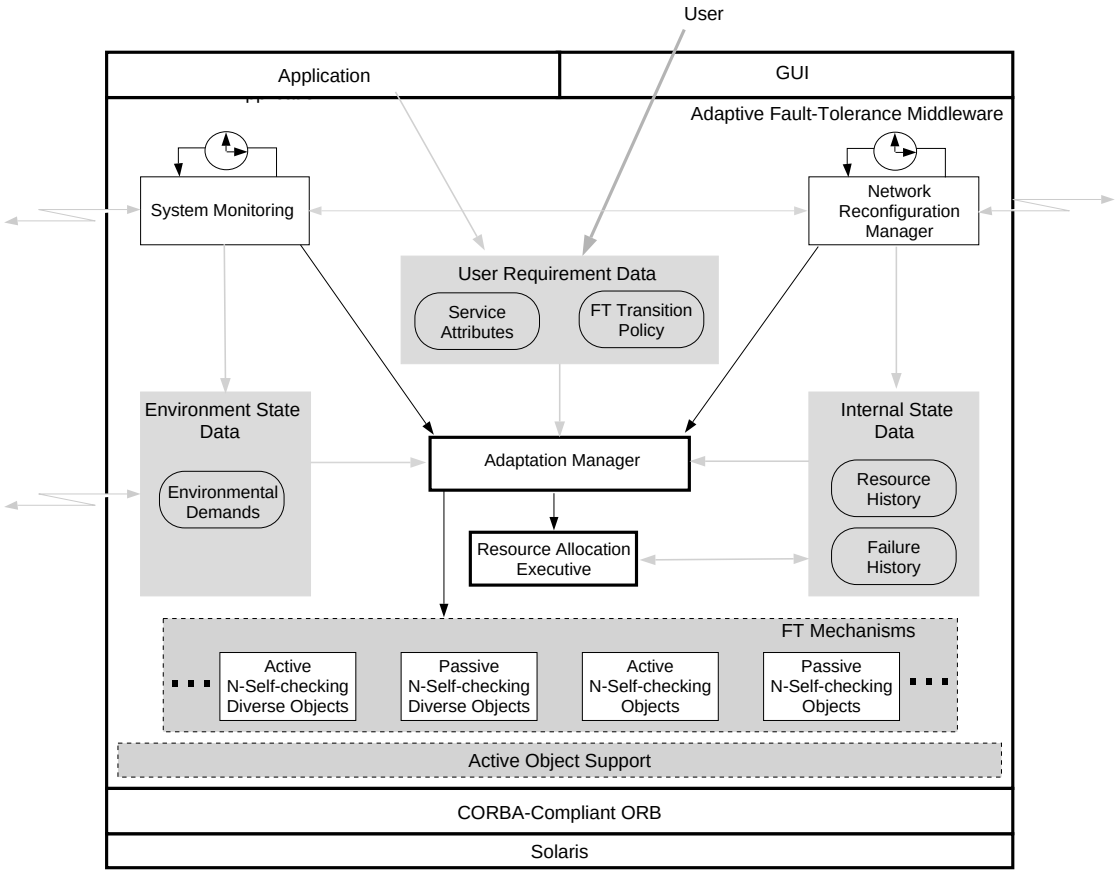

Figure 2: AFTM System Architecture

c) A concurrency constraint, which prevents conflicts between time-triggered methods and service methods, is incorporated.

d) Data members of an RTO may have limited valid duration. The data becomes invalid when its valid duration passes.

We believe the RTO support capability will lead to a more accurate and more responsive implementation of the both AFTM components and the application components.

The adaptation mechanism employed by the AFTM middleware utilizes the following three databases:

(i) *Environmental Database*: This database maintains recent changes in the environmental conditions and demands. It may be modified by the sensors interfacing with the environment. Moreover, the AFTM components responsible for monitoring and diagnosis (such as the System Monitoring component) may also update this database based on the recent diagnosis knowledge.

(ii) *Internal State Database* : This database maintains the recent failure history as well as the history of recent behavior of system resources. Network Reconfiguration Manager (NRM) and Resource Allocation Executive (RAE) update this database when an anomalous behavior of a system resource is diagnosed.

(iii) *User Requirement Database*: The user informs the AFTM middleware about the specific characteristics and requirements of the application by providing (1) application-specific adaptation policies (if they exist), and (2) application attributes such as acceptable reliability and performance characteristics of each application task.

The Adaptation Manager (AM) component selects the most suitable fault-handling and resource-allocation modes of the system based on the current contents of these three databases. The adaptation decision made by AM is forwarded to the Resource Allocation Executive (RAE) component to enforce the changes by reallocating available resources.

In order to realize an effective adaptation mechanism, any change in the health status of various software and hardware components of the cooperating distributed AFTM middleware should be known by the Adaptation Manager within an acceptable delay. The Network Reconfiguration Manager (NRM) component is responsible for the fast detection of any anomalous behavior of software and/or hardware components of the application, or AFTM. NRM also reconfigures the network when a permanent malfunction of a resource is diagnosed.

The System Monitoring (SM) component provides the user with the current status of application components. It is mainly responsible for monitoring the health status of active objects running in the nodes in the distributed system. This system monitoring facility enables the user to selectively monitor the health-status of various objects and instruct AFTM to take an appropriate action when the system enters an unexpected abnormal state.

As depicted in Figure 2, the AFTM middleware provides a variety of fault-tolerant execution modes from which the most suitable execution mode can be selected for each application task based on the task requirements and resource availability. These fault-tolerant execution modes will be discussed in more detail in Section 5.

4. Scalable Implementation of AFTM

The AFTM middleware should be scaleable such that it can be efficiently used in both LAN and WAN environments. Such a scaleable middleware can be realized by maximizing the autonomy of each LAN domain in the adaptation decision process. We envision potential applications such as distributed military planning applications as a collection of cooperative tasks each of which has its own specific timing and reliability characteristics. It is also beneficial to assume that each task will be entirely assigned to a LAN domain for the execution. In other words, if a task should be executed in replication, then all of its replicas should be executed in the same LAN. This localization of task replicas will not only lead to a more efficient AFTM implementation, but also facilitate faster anomaly detection and recovery. Figure 3 depicts the adopted system architecture.

As depicted in Figure 3, each LAN domain consists of a leader node and one or more follower nodes. The leader node in a LAN domain decides LAN-wide adaptation decisions by running the AM and RAE components (as shown in Figure 2). However, the AFTM middleware guarantees that all healthy follower nodes in a LAN domain have up-to-date information in their local

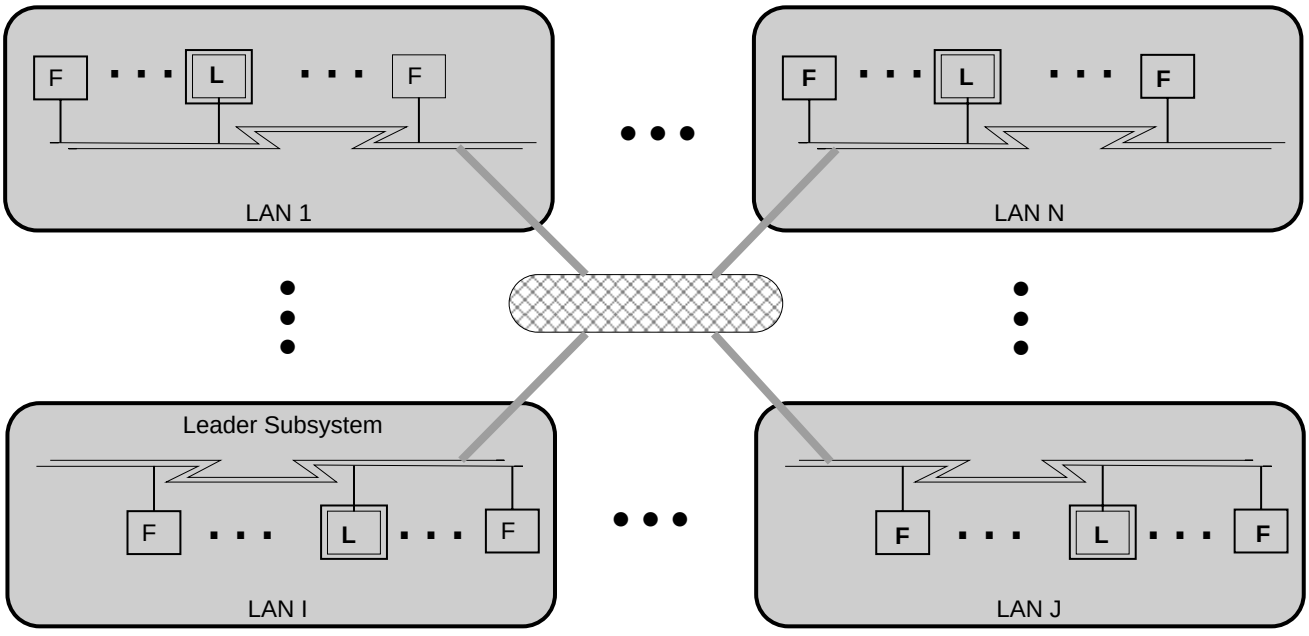

Figure 3: Adopted System Architecture

databases, so that if the leader node fails, then one of the follower nodes can become the new leader node within an acceptable time interval.

However, if the local leader node in one LAN domain recognizes that the domain is not capable of executing a task with full reliability and timing requirements, then the leader node may request the leader LAN domain (a specific elected LAN) to relocate the task into another capable LAN domain for execution. The leader LAN domain maintains information on resources and capabilities of all LAN domains. An efficient election of electing a new leader domain must be carried out, when the elected LAN domain is unable to carry the leading role.

We believe that this hierarchical leader/follower approach for adaptive resource allocation and reconfiguration will lead to a highly scalable and survivable system. Moreover, the performance of the adopted hierarchical structure in fault-free scenarios will be much higher than that of other conceivable (fully-decentralized or fully-centralized) alternatives.

5. Fault-Tolerant Execution Modes in AFTM

A fundamental issue in selecting candidate fault-tolerant execution modes, to be included in the AFTM middleware, is to identify the non-negligible types of anomalies in the selected application domains and the selected hardware and software platforms. A minimal set of fault-tolerant execution modes should then be selected to facilitate sufficient tolerance capabilities for the identified fault types while keeping the mode-transition manageable. Since errors and imperfections in complex (commercial or military) software cannot be ignored [11], the AFTM middleware should ideally support execution modes for tolerating both software and hardware failures.

Our selected fault-tolerant execution modes are highly influenced by *the Adaptable Distributed Recovery Block* (ADRB) concept discussed in [1,3]. The *Distributed Recovery Block* (DRB) scheme [12] is a task-level fault-tolerance scheme and executes different versions of the application task in different hardware nodes such that an occurrence of software or hardware failures in one node can be tolerated by using the result of other healthy nodes. The DRB scheme was extended in [13] to provide object-level real-time fault-tolerance (named primary shadow RTO.k replication - PSRR). We integrated ADRB and PSRR in our fault-tolerant execution modes which is shown in Figure 4

As depicted in Figure 4, the supported execution modes include the following three fault-tolerant execution modes:

- RTO-SRB: Under the RTO-SRB scheme each method of an RTO can be structured as a recovery block [14] and thus RTO-SRB provides rollback and recovery mechanism for recovering from non-crash failures.

- PSRR(RTO-DRB): PSRR applies the DRB concept for each important method of an RTO and thus provides parallel execution of each important method of an RTO. Therefore, the latency for any hardware and/or software failures can be kept very short.

- RTO-EH: Under this scheme, each method of RTO is equipped with an acceptance test and exception handler. If an output of a method fails the acceptance test, then the associated exception handler is invoked.

Table 1 summarizes various characteristics of the three fault-tolerant execution modes. As the node availability and the execution deadlines change, AFTM may switch among these three execution modes.

Table 1: Characteristics of FT Execution Modes

RTO-SRB	PSRR (RTO-DRB)	RTO-EH
Backward recovery	Forward recovery (Parallel execution of application)	Forward recovery
Relatively long recovery latency	Short fault-recovery latency	Short fault-recovery latency
No hardware redundancy	Hardware redundancy	No hardware redundancy
Dual versions may be required	Dual versions may be required	Exception handler is required

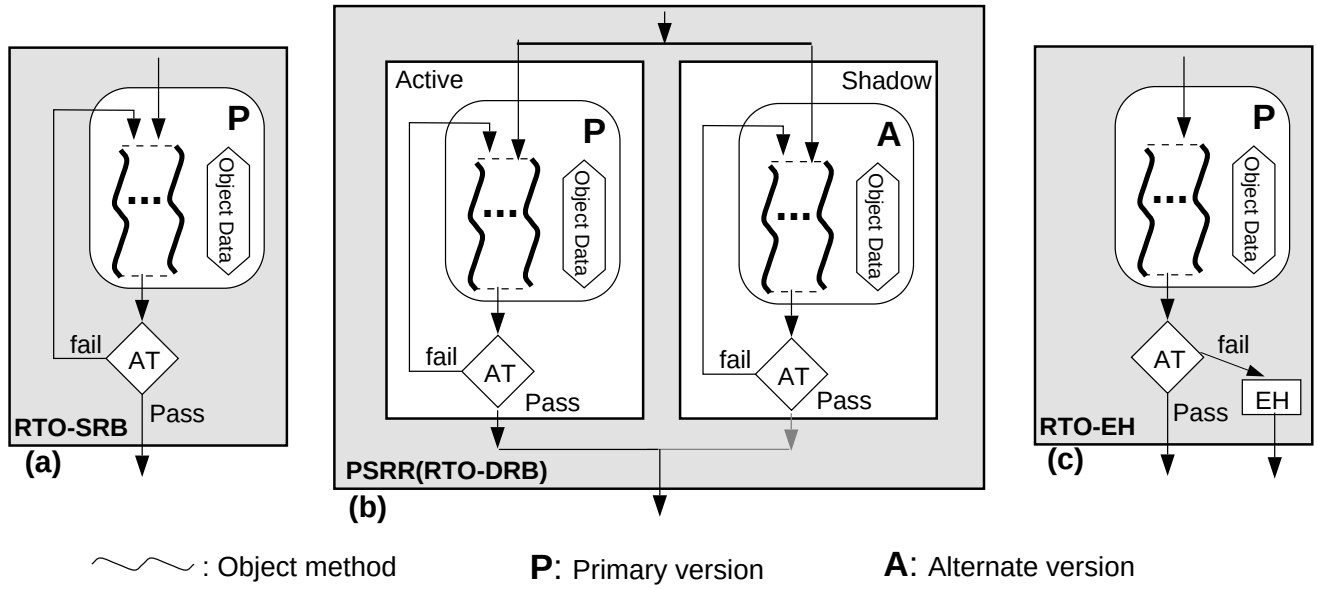

Figure 4: Major Fault-Tolerant Execution Modes

6. Conclusions

This paper provided an abstract architecture of a middleware under development. The architecture was devised to support efficient fault-tolerant execution of object-oriented real-time distributed applications.

The AFTM middleware is intended to possess additional capabilities beyond those which past AFT development efforts intended to realize. First, the AFTM components being developed are CORBA-compliant and will run on an open system platform (Solaris). This ensures the applicability of AFTM to a wide range of military, as well as commercial, applications. Secondly, the user of AFTM is given the facility for dynamically (during both application initiation and execution periods) interacting with the middleware to (i) enforce application-specific adaptation policies, and (ii) manually direct the adaptation process, if the system enters an unexpected state which cannot be handled by the AFTM mechanisms.

In order to evaluate its effectiveness, the AFTM middleware will be integrated with a distributed military planning application. We also plan to validate the design and implementation of the AFTM middleware by use of a user-friendly fault-injection subsystem. The graphical fault-injection subsystem will simulate the changes in both the environment and the internal states of the system and will facilitate an cost-effective evaluation of various capabilities of the AFTM middleware.

Acknowledgments: This work was supported in part by Rome Laboratory, U.S. Air Force under Contract F30602-96-C-0060 from.

7. References

[1] Kim, K. H., and Lawrence, T., "Adaptive Fault-Tolerance in Complex Real-Time Distributed Applications", *Computer Communication*, Vol. 15, No. 4, May 1992, pp. 243-251.

[2] Gupta, B. et al., "Adaptive Fault-Resistant System (AFRS)", *Rome Laboratory USAF, Technical Report* RL-TR-95-3.

[3] Goldberg, J., et al., "Adaptive Fault Resistant System", *SRI International, Technical Report* SRI-CSL-95-02.

[4] Bihari, B., and Schwan, K., "Dynamic Adaptation of Real-Time Software", *ACM Transactions on Computer Systems*, Vol. 9, May 1991, pp. 143-174.

[5] Hurley, P., and Dussault, J. L., "The Development and Assessment of An Adaptive Fault Resistant System (AFRS)", *Proc. of Second International Command & Control Research & Technology Symposium*, Warwickshire, UK, Sept., 1996.

[6] Booch, G., *Object Oriented Analysis and Design with Applications*, Redwood City, CA: Benjamin/Cummings Publishing, 2nd ed., 1994.

[7] Object Management Group*, The Common Object Request Broker: Architecture and Specification*, Revision 2.0, 1995.

[8] Shokri, E., and Hecht, M., "Adaptive Fault-Tolerance for Autonomous Spacecraft"*, NASA SBIR Phase I, Final Report*, SoHaR Inc., Beverly Hills, CA, June 1996.

[9] Hecht, H., and Crane, P., "Rare Conditions and Their Effect on Software Failures", *Proc.* 1994 *Reliability and Maintainability Symposium*, Anaheim, CA, Jan. 1994, pp. 334-337.

[10] Kim, K. H., and H. Kopetz, "A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials*", Proc. IEEE Computer Society's 1994 International Computer Software and Applications Conference*, November 1994, Taipei, pp. 392-402.

[11] Hecht, M., and Fiorentino, E., "Causes and Effects of Spacecraft Failures"*, Quality and Reliability Engineering International,* Vol. 4, No. 1, Jan. 1988, PP. 11-19.

[12] Kim, K. H., "*The Distributed Recovery Block Scheme*", Ch. 8 in M. R. Lyu ed., '*Software Fault Tolerance'*, John Wiley & Sons, Chichester, 1995, pp. 189-210.

[13] Kim, K. H., and Subbaraman, C., "Fault-Tolerant Real-Time Objects"*, Communications of ACM,* Vol.40, No. 1, Jan. 1997, PP. 75-82.

[14] Randell, B., "System Structure for Software Fault Tolerance", IEEE Transactions of Software Engineering, Vol. SE-1, No. 5, June 1975, pp. 220-232.