International Journal of Software Engineering and Knowledge Engineering Vol. 8 No. 3 (1998)

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-management 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 presents an architecture framework of an Adaptive Fault Tolerance Management (AFTM) middleware using a CORBA-compliant object request broker resting on the Solaris open system platform. The paper also discusses approaches which have been tested through an AFTM prototyping effort.

1. Introduction

The new generation of complex mission-critical applications (such as military planning and monitoring applications) is 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 changes in the environment, the application demands, and the system resources. Therefore, the capability called Adaptive Fault Tolerance [1] (AFT) is an essential feature for emerging highly complex mission-critical systems.

There have been several research and development efforts on adaptive fault tolerance carried out in laboratories [1,2,3]. The main emphasis in 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 remain to be explored. Moreover, most of these demonstration or prototype systems were implemented in non-standard or non-real-time hardware/software platforms, making porting to other hardware/software environments a difficult task. There remains a need for developing an AFT management subsystem that possesses the following important characteristics [4]:

- The AFT subsystem should be implemented in open standard hardware/software platforms,
- It should act as an interface between the application and the operating system kernel (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,
- It should support a set of *generic* fault tolerance mechanisms and adaptation policies that can be suitably customized for a wide range of distributed applications.

Considering the complexity of developing such a middleware, it is imperative to use state-of-the-art software engineering methodologies. Recent software engineering experience has demonstrated that object-oriented design and programming techniques, 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 interface for distributed computing objects. The standard is called the Common Object Request Broker Architecture (CORBA) [5]. CORBA provides a standard specification for location-transparent object interactions among distributed client-server objects. A CORBA-compliant communication middleware is a promising base for implementing distributed applications, since it provides high-level 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 build an AFT management (AFTM) middleware on top of a CORBA support middleware. This paper presents the architecture framework of the AFTM middleware developed.

Section 2 starts with a discussion on the desirable attributes of an AFTM middleware. Section 3 discusses major components of the AFTM middleware. The implementation issues are then discussed in Section 4. The fault-tolerant execution modes supported by the AFTM middleware are discussed in Section 5. Section 6 presents a cursory description of the current implementation. Conclusions are in Section 7.

2. Desirable Attributes of an Adaptive Fault Tolerance Management Middleware

In this section, we first discuss essential attributes of an effective adaptive fault tolerance approach. Then additional attributes of the AFTM middleware imposed by the candidate application domains and development environments will be briefly presented.

2.1 Essential Core Attributes of the AFTM Middleware

A desirable AFT middleware should serve to enhance system reliability, performance, and survivability by offering the following capabilities:

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

- **Tolerate all non-negligible fault types:** An important step in designing a fault tolerance function is to identify all failures that have non-negligible occurrence probability. One complex question here is whether to include software failures in the set of failure modes to be explicitly dealt with. Since experience has shown that the possibility of software design faults cannot be ignored in large-scale applications [7], it is useful to design the AFTM such that it can easily incorporate a variety of provisions for tolerating software faults whenever application designers decide to make investments for tolerance of software faults.

- **Maintain quality-of-services under changes in the environment, system, and user profile:** Changes in the environment may manifest themselves 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 experiencing physical failures of hardware components. Moreover, in considering mission-critical command and control (C^2) systems, there are phases of an operation or mission when small delays in system response time can be tolerated (e.g., system on a station performing surveillance/monitoring), yet during other phases the system must respond immediately (e.g., direct control of a weapon system during an enemy engagement). 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 capacity of the system may also become stressed due to user/application demands. Operational situations that are both complex and diverse make it difficult to anticipate all the possible scenarios the system may encounter, and practical considerations (e.g., cost, size, power consumption, etc) always limit total system capacity. The adaptation mechanism must effectively respond to these possible changes within an acceptable time bound.

- **Support application-specific adaptation parameters**: The user and/or the developer of an application may provide application-specific parameters that affect the adaptation decisions. An effective AFTM should be able to accept these parameters during the execution of the application and factor them into the decision-making process.

- **Facilitate multiple levels of adaptation:** A suitable adaptation policy should consider various strategies (or modes) of adapting the system in response to anomalous behaviors (actual or anticipated). The adaptation can be envisioned, for example, in one of the following ways: (i) switching from one fault handling mode to another, (ii) modifying parameters of the currently used fault handling mode, or (iii) modifying service attributes.

- **Provide required services in an application-transparent fashion**: The AFTM middleware should be implemented as a layer separate from the application software layer and must have a simple and clear interface with the application in order to maximize its openness.

Figure 1 illustrates a candidate set of input parameters provided by the system designer, system components, and operators to facilitate realization of the above capabilities.

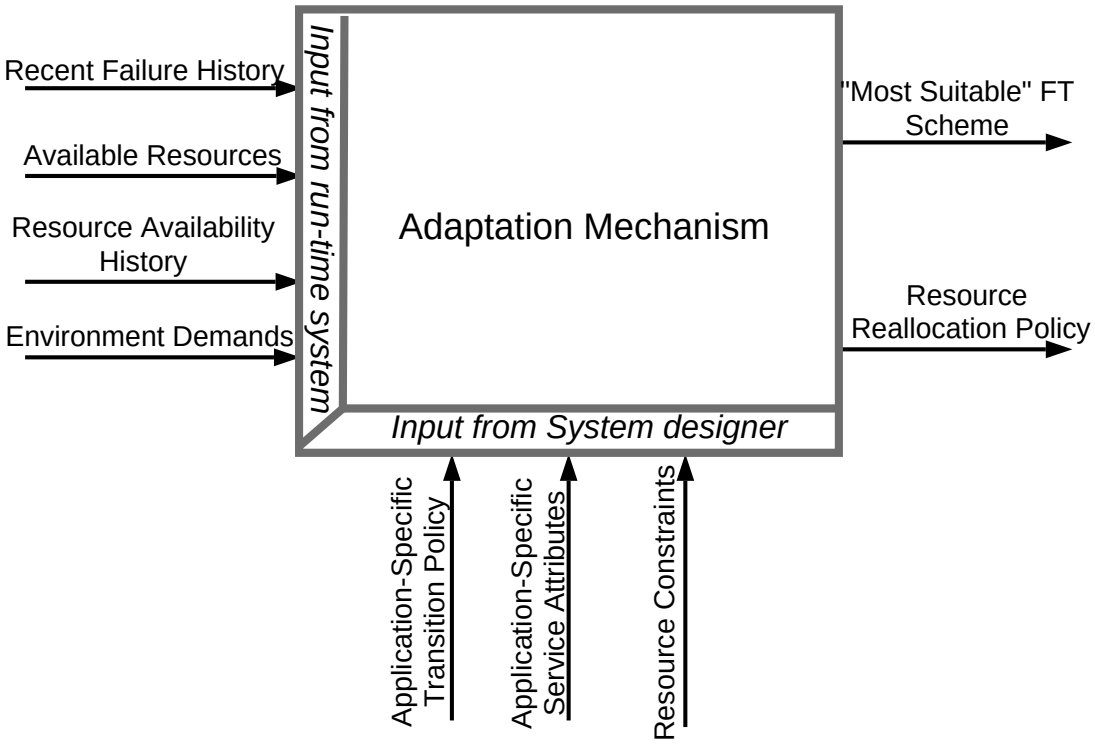

Figure 1: A candidate set of inputs to the adaptation mechanism

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 behavior according to the user commands. To facilitate this interaction with the user, the AFTM middleware should possess the following capabilities [4,6]:

- **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 the additional capability to govern the resource allocation when the system enters an unexpected anomalous state. To effectively interact with the AFTM middleware, the system user/administrator 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 that allows the user to probe into the system and request specific information about the application as well as various AFTM components.

- **LAN to WAN scalability:** In order to be applicable to a wide variety of applications, the AFTM middleware should be scalable and easily adaptable 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 runs on a network of Solaris workstations, which adhere to industry standards such as the Posix 1003.1b thread library. We have chosen Solaris over the more mature SunOS because (i) it facilitates application-level concurrency (multi-threaded programming) in an easy-to-control manner, and (ii) real-time performance, at least in an isolated intra-network environment, 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 popularity stems from the fact that (i) it makes development of distributed applications easier by providing high-level and simple inter-object communication services, and (ii) it facilitates 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 Time-Triggered Message-Triggered Objects (TMO's) (also called RTO.k objects) [8]. A TMO is an extension of conventional objects with the following added features:

a) Each TMO 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 requests (message-triggered methods).

b) Each method is associated with a completion deadline as well as output action deadlines.

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

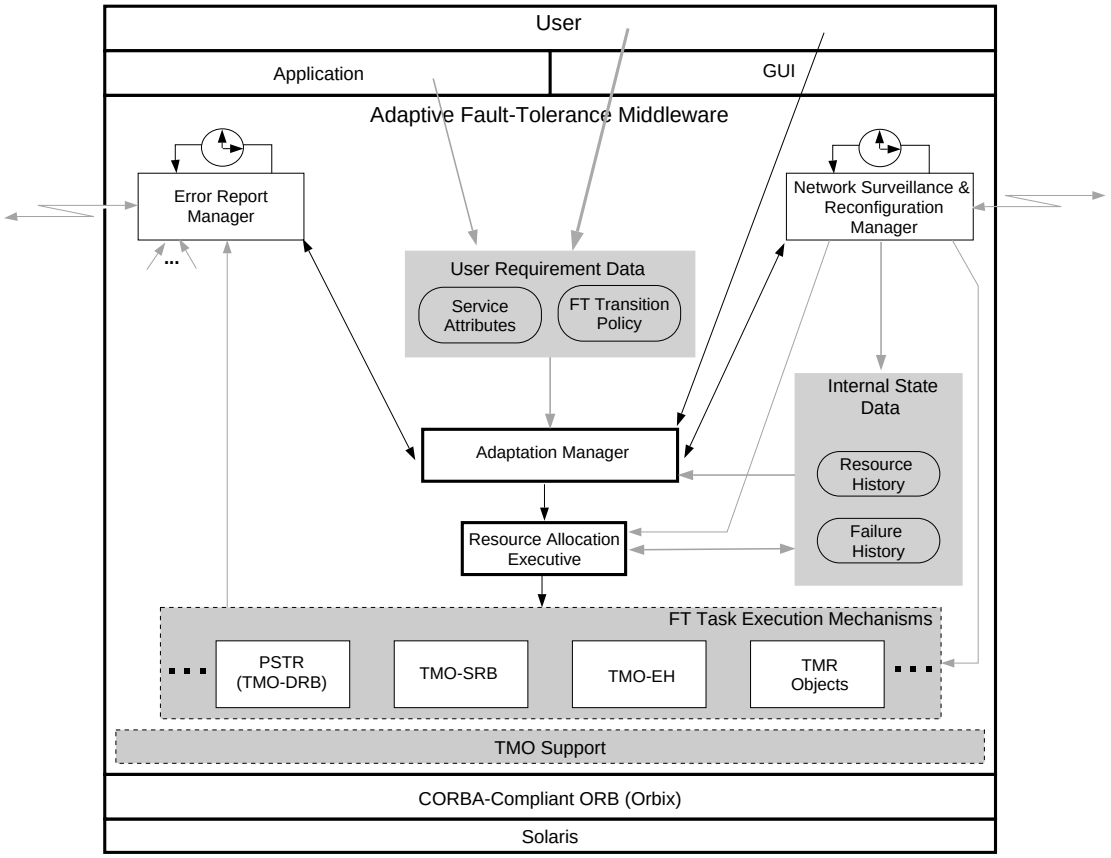

Figure 2: AFTM System Architecture

We believe the TMO structuring approach leads to a more accurate and responsive implementation of both AFTM components and the application components. 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 also believe that integrating the TMO structuring concept with the CORBA architecture will provide the system designer with the capability to design a variety of challenging distributed real-time computing applications as networks of CORBA-compliant TMO's.

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

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

(ii) *User Requirement Database*: The user and the application software inform 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 quality-of-service (QoS) requirements including 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 databases. The adaptation decision made by AM is forwarded to the RAE component which enforces the changes by reallocating available resources.

In order to realize an effective adaptation mechanism, any change in the health status of the software and hardware components of the distributed system must be made known to the Adaptation Manager within an acceptable time period (i.e., a minimal delay). The NSRM component is responsible for the fast detection of any anomalous behavior of hardware components and some low-layer software components. NSRM also orders RAE to reconfigure the network when a permanent malfunction of a resource is diagnosed. Therefore, NSRM's in distributed nodes cooperate among themselves.

The Error Report Manager (ERM) component is mainly responsible for monitoring the health status of application objects running in the nodes of the distributed system. It also receives reports on some errors detected by lower-layer software components such as the OS kernel, ORB, and fault tolerance task execution mechanisms. ERM forwards preprocessed error reports to AM. ERM's in distributed nodes cooperate among themselves.

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 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.

As shown in the architecture diagram of 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 adaptation-related information in their local 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. If a LAN domain designed as the leader can no longer carry out its leadership role, a new leader is elected.

We believe that this hierarchical leader/follower approach for adaptive resource allocation and reconfiguration is an effective way of realizing highly scalable and survivable systems. Moreover, during fault-free operations the performance of the adopted hierarchical structure is expected to be much higher than that of other conceivable (fully-decentralized or fully-centralized) alternatives.

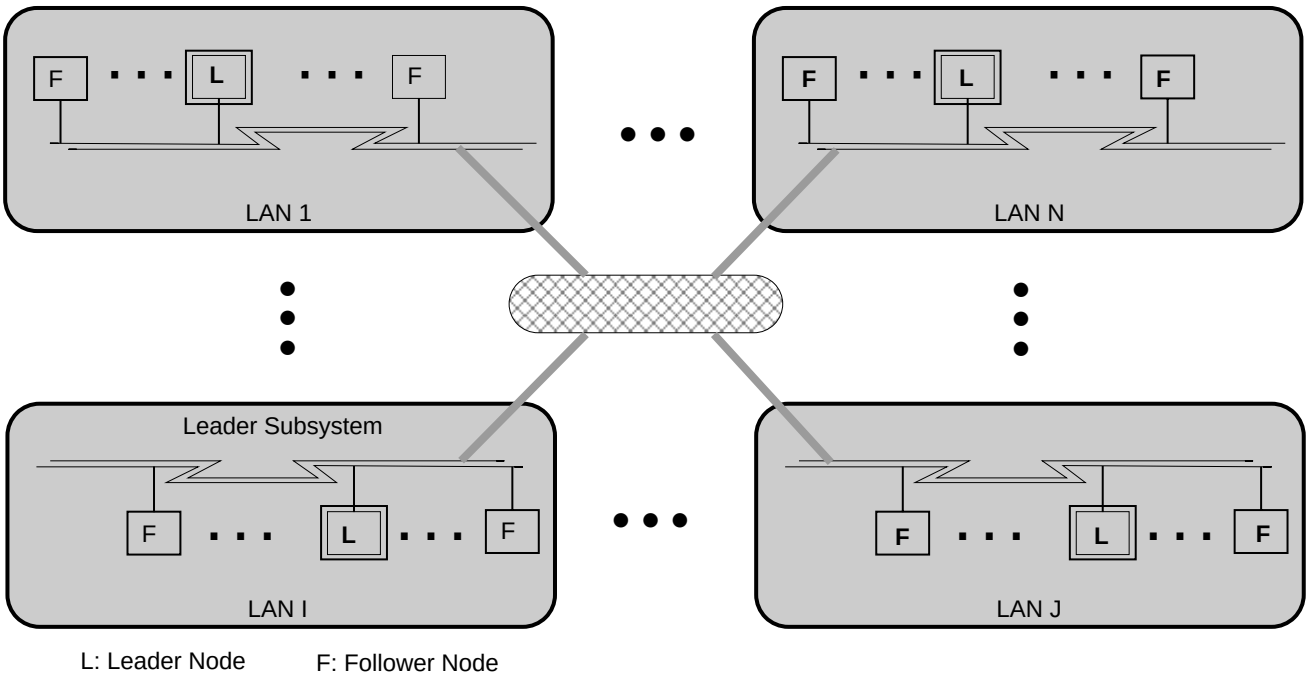

Figure 3: System Architecture

5. Fault-Tolerant Execution Modes in AFTM

A fundamental issue in selecting candidate fault-tolerant execution modes, to be provided by the AFTM middleware, is to identify all 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 for the identified fault types while keeping the mode-transition manageable. Since errors and imperfections in complex (commercial or military) software cannot be ignored [9], 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 [1,2]. The *Distributed Recovery Block* (DRB) scheme [10] 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 [11] to provide object-level real-time fault tolerance (named primary/shadow TMO replication - PSTR). We integrated ADRB and PSTR in our fault-tolerant execution modes that are shown in Figure 4.

As depicted in Figure 4, the supported execution modes include the following:

- TMO-SRB (Sequential Recovery Block): Under the TMO-SRB scheme each method of a TMO can be structured as a recovery block [12] and thus TMO-SRB utilizes rollback and recovery mechanism for recovering from non-crash failures.

- TMO-DRB (PSTR): Under the PSTR scheme, a TMO is replicated into a primary-shadow pair and thus for each important object method, its parallel redundant execution is realized with the method in the primary TMO and the same method in the shadow TMO. Therefore, the latency for any hardware and/or software failures will be very short.

- TMO-EH (Exception Handler): Under this scheme, each method of TMO is equipped with an exception handler. If an output of a method execution fails before or at the acceptance test, then the associated exception handler is invoked.

Table 1 summarizes the 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.

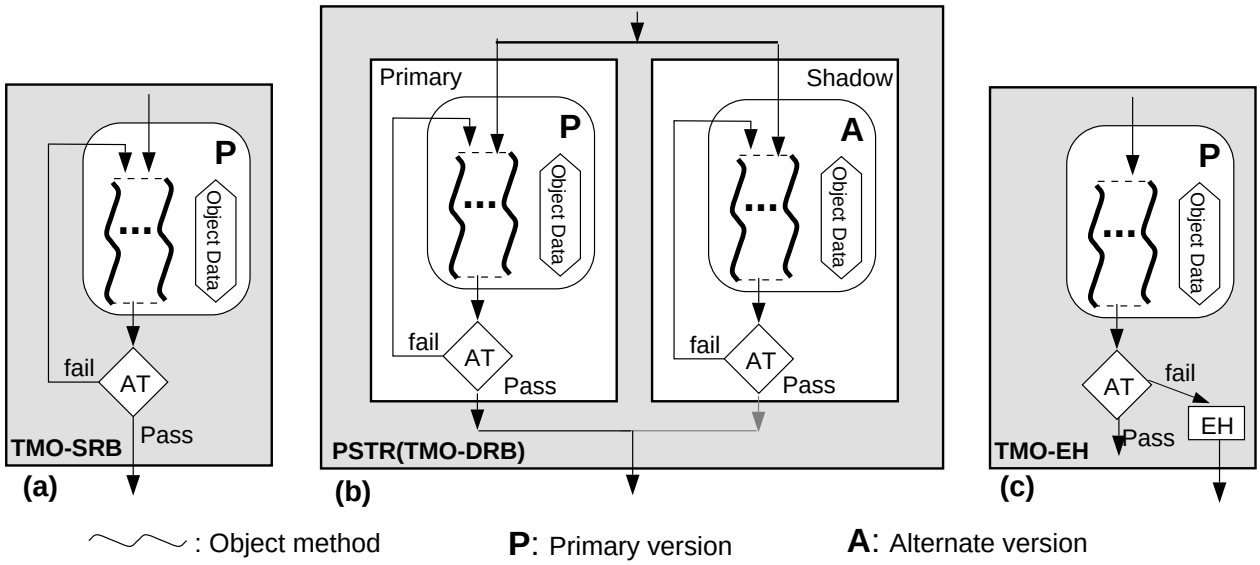

Figure 4: Major Fault-Tolerant Execution Modes

Table 1: Characteristics of FT Execution Modes

TMO-SRB	TMO-DRB (PSTR)	TMO-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

6. Specifics of the Current Prototype Implementation

A cursory description of some of the major aspects of the current prototype implementation of AFTM is provided in this section. More details on the implementation issues and lessons learned can be found in [13]. The hardware platform of the AFTM prototype is a network of Sun Sparc workstations running the Solaris 2.5 operating system. IONA's Orbix/MT (a multithreaded-ORB) with C++ binding was selected among available CORBA-compliant ORB's mainly because it provides a robust and efficient multithreaded inter-object communication capability. Moreover, OrbixWeb (Iona's Java binding ORB) was employed for developing various graphical user interfaces (GUI's) for users by taking advantage of Java's easy-to-use GUI development capabilities.

In order to separate concerns for reliability issues from those for functional issues and to simplify its implementation, the AFTM middleware was developed as a set of layers, each of which captures a specific functionality as depicted in Figure 5.

Due to space limits, we will not discuss the implementation issues for all these layers. Only the implementation issues in developing "TMO Management" and "Generic Fault Tolerance Services" layers are given a brief discussion here.

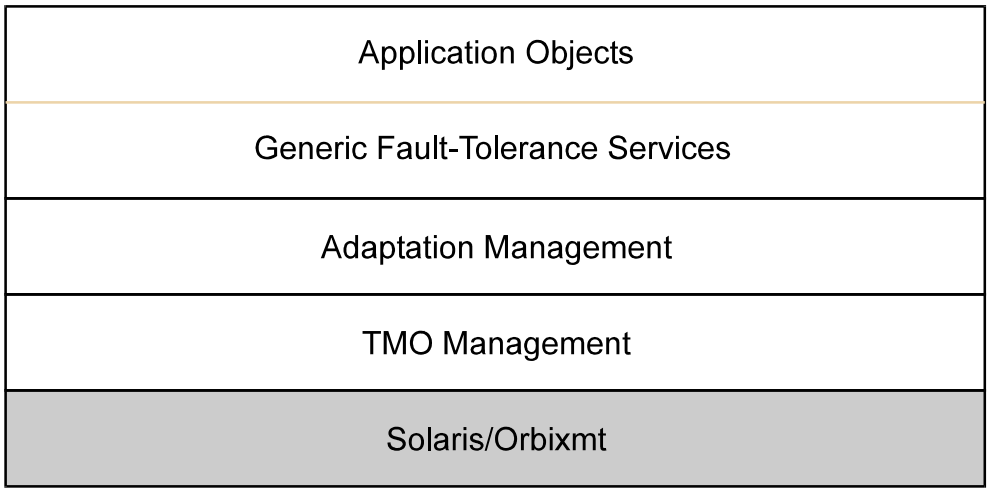

Figure 5: AFTM Layers

6.1 TMO Management Layer Implementation

The first layer of the AFTM is the TMO Management that mainly supports the execution of TMO's. A TMO registers its message-triggered and time-triggered methods with the TMO Management layer. This registration information is used for timely activation and execution of the registered TMO methods. As presented in Figure 6, the TMO Management layer is implemented using the following types of threads:

- *TMO Manager thread*: The TMO Manager thread handles scheduling and execution of all AFTM threads and all application TMO execution threads. It is implemented as a Solaris real-time thread and is periodically activated by a software watchdog timer. Since a Solaris real-time thread has the highest possible priority and is guaranteed to be executed immediately after its activation, the TMO Manager thread provides a degree of predictability for periodic execution of thread management functionality.

- *Server-Connection thread*: The server side of the Orbix daemon requires a thread to be introduced as an interface between the server and the daemon. The Server-Connection thread must be ready to handle all incoming and outgoing CORBA operation requests.

- *Service-Request threads*: Whenever a client TMO object attempts to call a remote TMO (or CORBA) method, it creates a Service-Request thread dedicated to make the remote method call and block until the result is returned. This is done for two reasons: (i) an object can concurrently conduct multiple service-call activities, and (ii) if the inter-node communication experiences a transient untimely behavior, the timing behavior of the client object will not be affected.

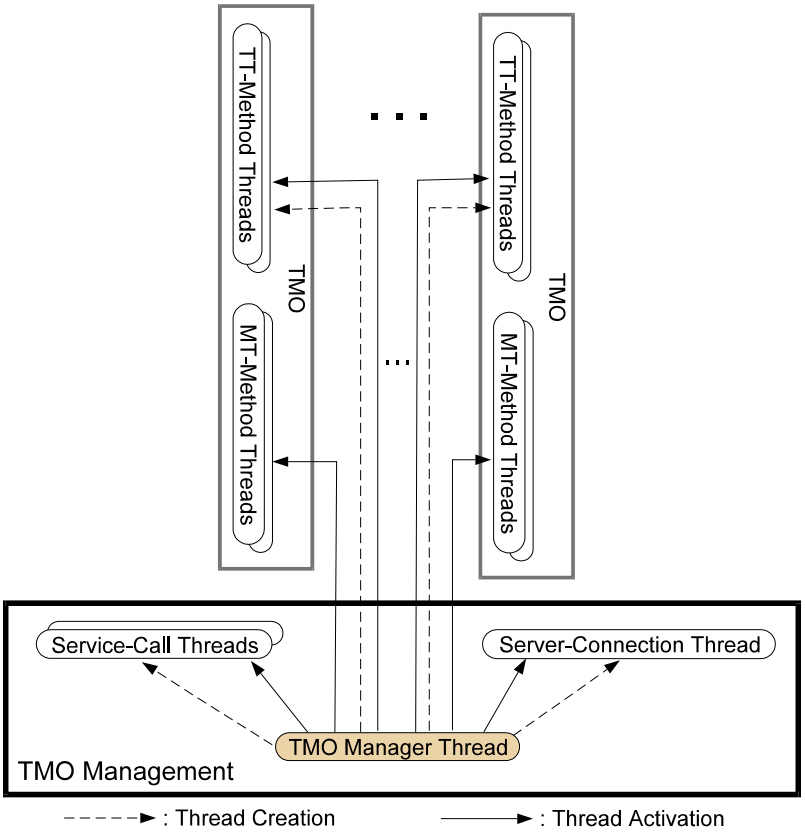

Figure 6: Threads in the TMO Management Layer

As briefly discussed in Section 3, message-triggered methods may be activated by a call from a client object. On the other hand, in order to enforce timely execution of TMO methods, the activation of any time-triggered or message-triggered method must be managed by the TMO Manager thread.

Therefore, the remote activation of a method call issued by Orbix must be intercepted by the TMO Manager threads. This interference is made possible by the "filter" facility introduced by Orbix. Orbix allows the application designer to provide additional code to be executed before or after the server program is executed [14]. This mechanism is termed filter. The filter code may enable the

application designer to enforce protective measures for nonfunctional characteristics such as security checks and debugging traps.

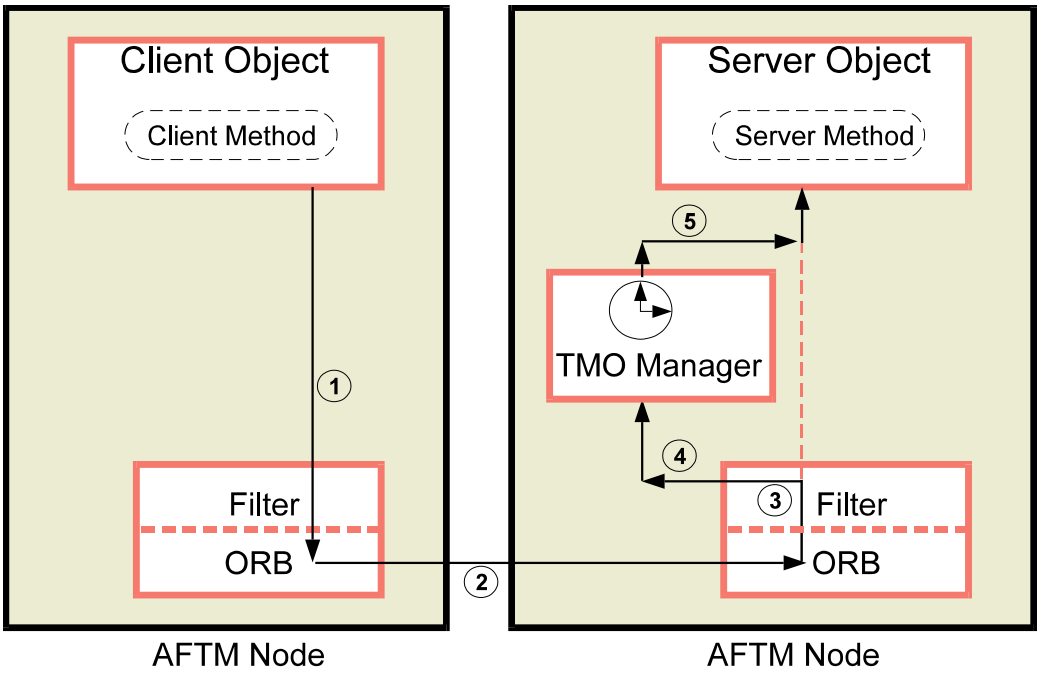

Figure 7: Anatomy of Remote Method Calls in AFTM

Figure 7 presents how the filter mechanism is used for remote method calls by the AFTM middleware. When an ORB receives a service request from a remote object, it executes the filter code which is modified to (i) create a thread for executing the service request, (ii) register the corresponding message-triggered method with the middleware, and (iii) suspend the created thread thereafter. In Figure 7, this series of activities are denoted by action number 4. The execution of the service request will then be initiated (based on its timely characteristics) by the TMO Manager thread in an appropriate point of time (action number 5 in the diagram). When the service execution is completed, the result will be forwarded back to the client object through the Server-Connection thread.

6.2 Implementation of Generic Fault-Tolerance Service Layer

Experience has shown that fault tolerance services can be designed and implemented in an application-independent (or application-transparent) manner offered by lower layers such as operating systems or middlewares. In other words, it is desirable to separate fault-management concerns from functional concerns of the applications. However, there are some fault-management elements which

are application-specific. For example, a non-trivial and realistic acceptance test (which checks the reasonableness of application outputs) varies from one application to another. It is thus useful to isolate application-independent elements from application-specific ones. Using object-oriented design and programming technology, it is feasible to capture the application-independent features as a base class such that each application can seamlessly inherit these features and add application-specific features in application-dependent derived classes [15]. By doing so, a complete separation of application-independent features from application-specific functions is achieved. In the AFTM middleware the Generic Fault-Tolerance Service Layer provides fault tolerance services as a set of abstract classes from which the application objects can be constructed.

The current prototype is a partial implementation in that each application task has a restrictive structure of a single service-method TMO. A full implementation planned for the near future will support applications structured as networks of general TMO's.

6.3 Implementation of the Graphical User Interaction

AFTM uses Iona's OrbixWeb Java binding for development of GUI's for the user/operator. The AFTM user interaction is supported by the following GUIs:

- ***AFTM System Monitor***: Provides the user with timely status information. The AFTM System Monitor window displays the current operating states of the registered application TMO's, the health status of the networked nodes, interaction between the primary and shadow partners, and newly joined nodes. The location and currently employed fault-tolerant execution mode of each registered application TMO, and a recent history of hardware and software configuration changes are displayed.
- ***System Requirement Specification Facility***: Provides a means for setting the initial system configuration and includes the window *Application Requirements* and the window *Environmental Conditions*. The Application Requirements window allows the user to set candidate hosts as well as timing requirements and applicable fault-tolerant execution modes for each registered application TMO. The Environmental Conditions window displays the best execution configuration for each TMO under current environmental conditions.
- ***Manual System Control***: Allows the user to provide specific adaptation commands to the system.

7. Conclusions

This paper provided an abstract architecture of a middleware facilitating adaptive fault tolerance. The architecture was devised to support efficient fault-tolerant execution of object-oriented real-time distributed applications. The architecture was partially validated through a prototype implementation.

The AFTM middleware reported here represents advances in the state of the art in several aspects. 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 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 is currently being integrated with a distributed command-control application. We also plan to validate the design and implementation of the AFTM middleware more thoroughly by use of a user-friendly fault-injection subsystem. The subject of AFTM middleware is still a relatively young one and much more research, especially application experiments, is needed to develop ways of exploiting the full potential of the adaptive fault tolerance.

Acknowledgments: This work was supported in part by Rome Laboratory, U.S. Air Force under Contract F30602-96-C-0060, and by DARPA under Contract N66001-97-C-8516.

8. 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. (An earlier version appeared in the *Proc. 2nd IEEE Computer Society's Workshop on Future Trends of Distributed Computing Systems*, Cairo, Egypt, Sept. 1990, pp. 38-46)

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

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

[4] 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, pp. 256-267.

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

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

[7] 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.

[8] Kim, K. H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No. 8, August 1997, pp. 62-70.

[9] 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.

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

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

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

[13] Shokri, E., Crane, P., and Kim, K H., "Architecture of ROAFTS/Solaris: A Solaris-Based Middleware for Real-Time Object-oriented Adaptive Fault Tolerance Support", *SoHaR Internal Technical Report*, December 1997.

[14] Orbix Programming Guide, *IONA Technologies*, Dublin, 1995.

[15] Shokri E. and Tso, K. S., "Development of Software Fault-Tolerant Applications with Ada95 Object-Oriented Support"*, The National Aerospace and Electronics Conference*, Dayton, OH, May 1996, pp.519-526