

# Adaptive Fault-Tolerant CORBA Components

Fábio Favarim<sup>1</sup>, Frank Siqueira<sup>2</sup>, Joni Fraga<sup>1</sup>

<sup>1</sup>*Departament of Automation and Systems and* <sup>2</sup>*Departament of Computer Science*  
*Federal University of Santa Catarina – Florianópolis, SC 88040-900 – Brazil*  
fabio@das.ufsc.br, frank@inf.ufsc.br, fraga@das.ufsc.br

## Abstract

Component-based software development allows developers to compose applications using software parts, called components. This approach represents an important step towards reducing development cost and time, and allows the creation of automated tools for software development. However, current applications have quality of service (QoS) requirements, and the existing component-based technologies provide limited support for QoS. In particular, fault tolerance is provided only in the form of persistence services, which enable the recovery of faulty components without loss of state information. This paper presents AFT-CCM (Adaptive Fault-Tolerance in the CORBA Component Model) – a model for building component-based applications with QoS requirements related to fault tolerance. AFT-CCM is based on CCM (CORBA Component Model), proposed by the OMG (Object Management Group), which extends the CORBA object model in order to introduce a new abstraction – the component. The AFT-CCM model allows the application user to specify QoS requirements that will be interpreted in order to adapt the configuration of replicated components. Characteristics such as the number of replicas and the replication technique will be defined aiming to fulfill the fault tolerance requirements specified by the user. This process is accomplished at execution time employing a set of CCM components that deal with the non-functional aspects of the application. The characteristics of this model, a prototype implementation, and an analysis of its functionality in the face of other proposals found in the literature are described along this paper.

## 1. Introduction

The component-based software development approach has been pointed out as a new milestone in the history of software development. By composing applications from pre-existing self-contained components with well defined interfaces, the cost of the software development process can be reduced sharply. In addition, the use of replaceable software components simplifies the implementation and the maintenance of complex applications [1]. Commercial component-based software development models, such as Enterprise JavaBeans (EJB), developed by Sun Microsystems [2], Microsoft .NET [3] and the CORBA Component Model (CCM) [4], are being used widely and have shown improvements in the software development and maintenance process.

Current software systems are becoming even more distributed and operating in highly dynamic environments as the Internet. In this way, distributed applications with fault-tolerance requirements are difficult to build and maintain. Currently available component models provide limited support for fault-

tolerance in the form of mechanisms for data persistence. Consequently, if an application component or a host machine does not function properly, the whole distributed application may fail.

In this paper, we use dynamic configuration of components to provide adaptive fault-tolerance [5][6], which allows a component-based distributed application to behave as expected despite possible component and machine faults. Different levels of dependability or availability can be provided by using different replication techniques and an appropriate number of components replicas.

The AFT-CCM model (Adaptive Fault-Tolerance in the CORBA Component Model) shows how fault-tolerance can be achieved with complete transparency for the application without changes to the component model and its implementation. The AFT-CCM model is formed by software components that are responsible for implementing fault-tolerance techniques, defining and controlling the behavior of a replicated service. The model integrates QoS specification mechanisms that guide the definition of the configuration of replicated

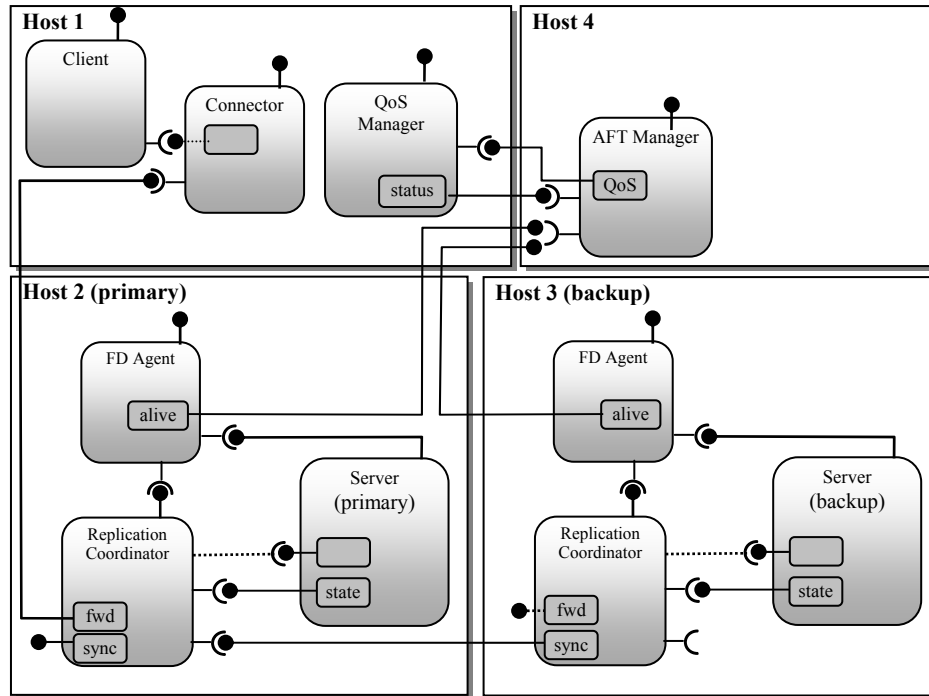


Figure 1: The AFT-CCM Model

services. In this way, different levels of quality of service (QoS) can be specified, aiming to provide different fault-tolerance requirements. The AFT-CCM has fault detection mechanisms that identify the need for adaptation, triggering changes in the replication techniques and in the application configuration aiming to enforce the QoS requirements specified by the user. This process is accomplished without compromising the performance and the stability of the application.

This paper is organized as follows: section 2 presents the AFT-CCM model; in section 3 the prototype implementation of this model is presented; section 4 analyses related proposals found in the literature and in section 5 we present our conclusions and perspectives for future work.

## 2. The AFT-CCM Model

The adaptive fault-tolerance model profits from the flexibility that arrives from the adoption of a component-based technology. In this project we have chosen to use the CORBA Component Model [4] due to its rich communication semantics and to its openness, which allows it to be used in an heterogeneous environment, in which different programming languages, middleware and operating systems may coexist.

The proposed model allows the application programmer to specify quality of service (QoS) requirements, defining desired levels of availability of the service, and the execution support defines a configuration and employs a replication technique aiming to fulfill such requirements. The support provides the non-functional code that monitors the occurrence of faults in the application and adapts its behavior in order to maintain the desired level of QoS specified by the user.

Figure 1 illustrates the different components that form the execution support of the AFT-CCM model. Essentially, these non-functional components configure and monitor application components, replicating components and using a replication technique in order to achieve the desired levels of reliability. In the example shown in Figure 1, a component identified as the primary server is located on host 2; its replica (the backup server) is placed on host 3. Each replication technique has a protocol for maintaining replica consistency. In our model, this protocol is implemented by a component: the *replication coordinator* (Figure 1). In this way, when the passive replication technique<sup>1</sup> is used, replicas states are synchronized by the replication coordinator using mechanisms of state transfer. The replication technique can be easily changed at run-time

<sup>1</sup> In the passive replication technique [7], after executing a request from a client, the primary server must send a copy of its state to synchronize backup replicas.

by replacing this component with another coordinator that implements a different replication technique.

The client is able to access component services through a *connector*, which hides changes in the configuration of the application from the client. For example, the exchange of replicas, where the backup replica would replace the primary, does not affect the behavior of the client. If the primary replica fails, the connector redirects calls for the backup replica of the server.

For each component with fault-tolerant requirements, an *adaptive fault-tolerance manager* (AFT manager) is created. The AFT manager defines the current configuration, based on the QoS requirements demanded for the application and on the frequency of partial failures. A reconfiguration of the application is started when monitoring data collected by the AFT manager shows that the current configuration is not appropriate for maintaining the QoS requirements of the application – i.e., it may be in lack or in excess of resources (replicas) or using a replication technique weaker or stronger than necessary.

Faults are monitored using the *fault detection agents* (FD agents, in Figure 1). Each component replica is bound to a FD agent, which is the responsible for sending monitoring data to the AFT manager. By using the FD agents, two levels of faults can be detected: machine faults – in case the FD agent stops responding – and component faults – i.e., when the component stops answering calls from the FD agent.

The *QoS manager* allows the user to specify his QoS requirements. Through the QoS manager different levels of QoS related to the reliability of the application can be specified. These requirements are passed on to the AFT manager, which interprets them and defines a proper configuration for the application – i.e., the number of replicas and the replication technique that will be used. The AFT manager tries to keep the requested level of QoS with the resources currently available. The AFT manager changes the configuration of the system when it detects that the requirements are not being fulfilled. QoS requirements can be changed at any time through the QoS manager. Any change in the requirements is informed to the AFT manager, which may need to reconfigure the application in order to enforce the new QoS requirements. The QoS manager also informs to the user the current configuration of its application – i.e., the number of replicas, their location, the replication technique being used by AFT-CCM and statistics about the frequency of faults in the application.

### 3. Implementation

The AFT-CCM model, which was described in the previous section, is composed of a set of software components that were implemented in a prototype. This prototype implementation was built using OpenCCM version 0.2 [8], which is a partial implementation of the CCM specification. OpenCCM was chosen because it was the first open source implementation of CCM. The version 0.2 of the OpenCCM runs on three different ORBs. In this project we are using ORBacus version 4.0.5 [9].

The prototype implementation of the AFT-CCM model provides three different replication coordinators: a void coordinator, which does not implement any replication technique, a second one that implements the passive replication technique, and a third that implements the semi-active<sup>2</sup> technique. Active replication was not implemented, because it requires group communication mechanisms that are not provided by ORBacus.

Some of the components required us to adopt creative solutions. For example, the connector and the replication coordinator use CORBA's dynamic skeleton interface (DSI) and dynamic invocation interface (DII) respectively, in order to receive requests and redirect them to the replicated component without having to implement the component's interface. This dynamic invocation is represented in Figure 1 with a dotted line.

The state of the AFT manager is stored persistently, so that it can be restored in case of a machine crash. The QoS manager is responsible for monitoring the AFT manager's activity and relaunching it when it stops responding.

In order to allow that replication coordinators store and restore the state of component replicas, these components must implement the interface state, which has methods to set and get the state of a component.

Component monitoring is performed by the AFT manager on polling intervals defined by the user through the QoS manager. The AFT manager pings periodically the FD agent, which replies sending the current state of the components that it monitors. If the FD agent does not answer, a machine crash is assumed and the application may be reconfigured by the AFT manager. Component faults are detected by the FD agent by polling the component, and are informed to

---

<sup>2</sup> Semi-active replication implies that all replicas execute a method, but only one replica sends a reply to the client [10].

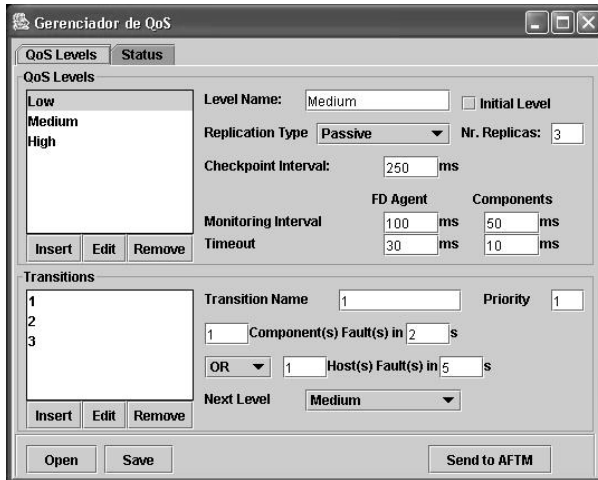


Figure 2: The graphical user interface of the QoS Manager

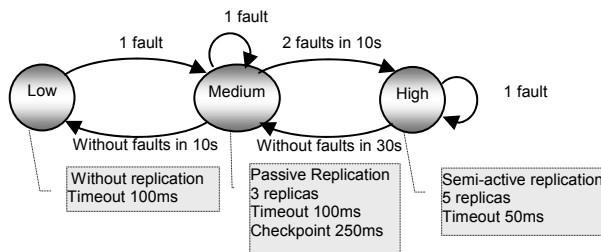


Figure 3: QoS levels

the AFT manager on the next monitoring period. Application reconfiguration may also be performed in this case, based on the levels of QoS defined by the user with the graphical interface of the QoS manager, which is shown in Figure 2. By using this graphical interface, the user can define QoS levels and the conditions that trigger transitions between them, forming a state machine like the one shown in Figure 3.

#### 4. Related Work

Although the first works in the area of software components have appeared already in the 60s [1], the incorporation of this programming style to middleware is recent. Implementations of CCM are only beginning to appear. Additionally, despite the advantages of using software components to develop distributed applications, applications with requirements related to timeliness, fault-tolerance, load balancing, among others, do not find support for these requirements in the current component models. This is because the execution environment – the container – does not offer support to QoS requirements. The work presented in this paper is an effort towards using CCM to structure applications with redundancies managed by the execution support.

Some proposals that aim to provide support for QoS requirements of applications, in some cases using adaptive fault-tolerance, are described in the literature. The AQuA architecture (Adaptive Quality of Service Availability) [11] aims to supply adaptive fault-tolerance for distributed applications. AQuA allows applications developers to specify the desired levels of dependability, which are reached through the configuration of the system in accordance with the availability of resources and the faults occurred. AQuA uses QuO [12] to specify QoS requirements at application level, and the Proteus dependability manager [13] to configure the system in response to faults and availability requirements. Ensemble [14] is also used by AQuA in order to provide group communication services.

Despite having mechanisms to specify QoS requirements with equivalent functionality, QuO and AQuA require QoS requirements to be defined at compilation time, while AFT-CCM allows QoS requirements to be modified at execution time, taking advantage of the flexibility provided by the use of software components.

In [15] is presented the AFTM (Adaptive Fault-Tolerant Manager), which is an adaptive fault-tolerant middleware that uses a CORBA-compliant object request broker. The AFTM acts as an interface between the application and the underlying software layers that transparently monitors application behavior as well as resource availability, and adaptively reconfigures the system resources. The AFTM has two databases that are used in the adaptation process. The first database maintains the recent history of faults and behavior of system resources. And the second one has application requirements such as the acceptable reliability and the desired performance of each application task. The AFTM provides several execution modes, from which are selected the most suitable fault handling and resource allocation modes of the system based on the contents of its databases.

The AFTM provides an adaptive fault-tolerance support for applications based on CORBA objects, while AFT-CCM is aimed at applications based on CORBA components (CCM). AFTM employs fixed fault-tolerance strategies, while in the proposed model strategies can be defined according to the application requirements.

Chamaleon [16] is an adaptive infrastructure that provides different levels of availability through an architecture composed of ARMORs – Adaptive,

Reconfigurable, and Mobile Objects for Reliability. Chamaleon can select different combinations of ARMORs in order to provide different availability levels that can be introduced incrementally in the system. Although it uses composition mechanisms similar to the ones that exist in components models, Chamaleon does not adopt a standard components model such as EJB, .NET or CCM, which is used in this work.

In [17] is presented an approach for CORBA components replication. This approach uses interception objects that are responsible for capturing the invocations made to the component in order to trigger necessary actions for replication management. These interception objects have the same interface of the replicated component. This implies that for every new component that you want to replicate, it will be necessary to implement a new interception object with the same interface of the component. The AFT-CCM model uses a generic connector that is independent of the replicated component interface, so that it does not have to be modified to be used in different applications.

## 5. Conclusions and Future Work

The AFT-CCM model adds support for fault-tolerance to the CORBA Component Model by using a set of non-functional components that enforce and monitor the QoS requirements specified by the user, performing adaptation whenever necessary.

A prototype implementation of the model has been implemented. Preliminary performance measurements have shown that the model adds a small overhead to the application, with the great advantage of providing a much higher dependability. In the near future, we intend to evaluate the use of this prototype by using it for building distributed applications with fault tolerant requirements.

## References

- [1] Szyperski, C. *Component Software: Beyond Object-Oriented Programming*. ACM Press – Addison-Wesley Publishing Co. 1998.
- [2] Sun Microsystems. *Enterprise JavaBeans Specification*. v2.0. 2001. <http://java.sun.com/ejb/>
- [3] Microsoft. *Overview of the .NET Framework*. MSDN Library White Paper. 2001. <http://msdn.microsoft.com>
- [4] OMG. *CORBA Components*. OMG Document formal/02-06-65. 2002. <http://www.omg.org>
- [5] Hiltunen, M., Schlichting, R. Adaptive Distributed and Fault-Tolerant Systems. *International Journal of Computer Systems Science and Engineering*, 11(5):125–133. September, 1996.
- [6] Chen, W.-K., Hiltunen, M. A., Schlichting, R. D. Constructing Adaptive Software in Distributed Systems. *21st International Conference on Distributed Computing Systems*. April, 2001.
- [7] Budhiraja, N. et al. The Primary-Backup Approach. In: *Distributed Systems*. Mullender, Sape (Ed.). Addison Wesley. 2<sup>nd</sup> Edition. 1993.
- [8] Marvie, R., Merle, P., Vadet, M. *The OpenCCM Platform*. 2002. <http://corbaweb.lifl.fr/OpenCCM/>
- [9] Iona Technologies. *ORBacus for C++ and Java*, version 4.0.5. 2001. <http://iona.com>
- [10] Powell, D. (Ed.) *Delta-4 – A Generic Architecture for Dependable Distributed Computing*. ESPRIT Research Reports. Springer Verlag, 1991.
- [11] Cukier, M. et al. AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. In *17th IEEE Symposium on Reliable Distributed Systems*. 1998.
- [12] Zinky, J. A., Bakken, D. E., Schantz, R. E. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1):53–73. April, 1997.
- [13] Sabnis, C. et al. Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA. In *7th International Working Conference on Dependable Computing for Critical Applications*. 1999.
- [14] Hayden, M. G. *The Ensemble System*. PhD thesis, Cornell University. 1998.
- [15] Shokri, E. Hecht, H., Crane P., Dussault, J., Kim, K. An approach for Adaptive Fault-Tolerance in Object-Oriented Open Distributed Systems. In *3rd Workshop on Object-Oriented Reliable Distributed Systems*. 1997.
- [16] Bagchi, S. et al. The Chameleon Infrastructure for Adaptive, Software Implemented Fault Tolerance. In *17th Symposium on Reliable Distributed Systems*. 1998.
- [17] Marangozova, V. and Hagimont, D. An Infrastructure for CORBA Component Replication. In *1<sup>st</sup> IFIP/ACM Working Conference on Component Deployment*. 2002.