An Adaptive Fault-Tolerant Component Model

Joni Fraga[1], Frank Siqueira[2], Fábio Favarim[1]

[1]Department of Automation and Systems (DAS) and [2]Department of Computer Science (INE)
Federal University of Santa Catarina (UFSC), Brazil
fraga@das.ufsc.br, frank@inf.ufsc.br, fabio@das.ufsc.br

Abstract

This paper presents a component model for building distributed applications with fault-tolerance requirements. The AFT-CCM model selects the configuration of replicated services during execution time based on QoS requirements specified by the user. The configuration is managed using a set of components that deal with the non-functional aspects of the application. The characteristics of this model and the results obtained with its implementation are described along this paper.

1. Introduction

Current distributed software systems operate in highly dynamic and large-scale environments such as the Internet. Distributed applications are difficult to build and maintain, taking into account the complexity and the characteristics of these environments. This difficulty is increased further when the application has quality of service (QoS) requirements regarding timeliness and dependability.

The component-based software development approach, which is based on composing applications from pre-existing self-contained components with well-defined interfaces, aims to simplify the implementation and the maintenance of complex applications, such as distributed systems. By composing applications from pre-existing components, the cost and the duration of the software development process can be reduced sharply [1]. In addition, the use of replaceable software components that provide services through well-defined interfaces allow the developer to handle components as independent configuration units, bringing more flexibility to the implementation and maintenance process. The component-based software development approach can also been employed as a base for creating automated techniques for software production.

In the last few years, the concept of software components is being integrated to existing middleware technologies. Examples of these efforts are CCM (CORBA Component Model) [2], which was defined by CORBA 3.0 specification [3]; EJB (Enterprise JavaBeans) developed by Sun Microsystems, which is part of Java 2 Enterprise Edition [4]; and .NET, proposed by Microsoft, which is an evolution of the COM (Component Object Model) platform [5]. These middleware technologies provide limited support for fault-tolerance in the form of mechanisms for data persistence.

In this paper, dynamic configuration of components is employed to provide adaptive fault-tolerance, which allows a component-based distributed application to behave as expected despite possible component and machine faults. Different kinds of faults may occur as a consequence of the dynamic changes that occur in the system environment, and the adaptation of fault-tolerance mechanisms according to these variations aims to maintain the level of dependability and availability of the system. Different levels of reliability can be provided by using different replication techniques and an appropriate number of components replicas, allocating only the necessary resources in order to obtain the desired behavior from the application.

As a result, we have developed the AFT-CCM model (Adaptive Fault-Tolerance on the CORBA Component Model), which implements a model that employs adaptive fault-tolerance in a way completely transparent for the application. The AFT-CCM is formed by software components that are responsible for implementing fault-tolerance techniques, defining and controlling the behavior of replicated services. This paper presents a proposal to integrate support for QoS requirements to the component model. The QoS requirements specified by the user guide the definition of the replicated services employed in order to increase the application dependability. Different levels of QoS can be specified, aiming to provide different fault-tolerance requirements. The AFT-CCM has fault detection mechanisms that activate the adaptation, triggering changes in replication techniques and in the application configuration, aiming to maintain user-defined QoS requirements. This process is accomplished without compromising the performance and the stability of the application.

This paper is organized as follows: section 2 presents an overview of the component-based software development approach and describes the CORBA Component Model; section 3 presents the AFT-CCM model; in section 4 is presented the prototype implementation of AFT-CCM and the obtained results; section 5 analyses related proposals found in the literature; and section 6 presents our conclusions.

2. Software Components

Component-based software development consists in composing applications from pre-existent software entities, called components. This means that the software development process is centered on software reuse: building applications from components that have been tested during their development phase and have proven efficiency in other applications, implies in maintenance cost reduction. System maintenance, which is possible through component exchange, is facilitated because it targets only the necessary components.

Every component-based implementation must adopt a component model. This model defines the form in which the component interface must be published and the way to specify the methods and events that make its services available to the clients; interfaces define the access points to services offered by components, separating the specification from the implementation itself, and protecting users from implementation details. The component model must also provide guidelines for component creation and implementation [6].

Considering the existing middleware technologies available, the CORBA Component Model (CCM) is among the most promising ones. Since CCM is totally based on CORBA, it provides language and operating system independence, features that are not found in other component models, which are usually language (EJB) or platform dependant (.NET). CCM, described in the next section, was adopted in this work because of the comprehensive collection of services provided by CORBA [7] and due to the flexibility presented by its component model compared to other available technologies.

2.1 The CORBA Component Model

The CORBA Component Model (CCM) is an extension to the CORBA distributed object model. The component modeling, programming, packaging, deployment and execution stages described in the CCM specification organize objects in components, incorporating the characteristics regarded to component-based software development to the application development process defined by CORBA. Component interfaces are specified using CORBA IDL (Interface Description Language), which since version 3.0 was extended to allow component definition.

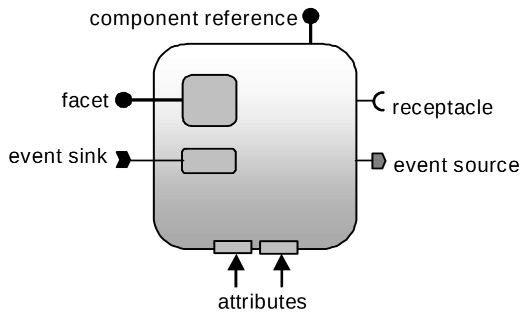


Figure 1. A CORBA Component

CORBA components have attributes and communication ports. Attributes are properties employed to configure component behavior. Communication ports are connecting points between components, through which they interact. Figure 1 illustrates the four kinds of ports defined by CCM specification [2]: facets, receptacles, event sources and event sinks[1].

Just like CORBA objects, components instances are characterized by interfaces and a reference. The component reference allows clients to browse through the ports of a component instance and connect these ports to their compatible ports – i.e. connect facets to receptacles with the same interface types, and connect source to sink ports with the same event type. Compatible ports can be connected at deployment time (static configuration) or during execution time, characterizing dynamic configuration.

CCM defines the *container*, a standard execution support for components, which allows transparent access to a set of common CORBA services for implementing non-functional aspects of an application. These CORBA services are composed by transaction, persistence, life cycle, security and event notification services. In other words, the container separates functional and non-functional aspects of the component. This way, application development is simplified, since the developer concentrates efforts on the functional part of the application.

Some programming facilities are defined in CCM to allow the description of the interaction between the functional and non-functional parts of applications. CCM

[1] Facets represent IDL interfaces through which a component receives requests to its services; receptacles are the connection points through which services provided by other components can be requested; and event sources and event sinks are ports that allow the exchange of events between components.

provides a Component Implementation Framework (CIF), which uses the Component Implementation Definition Language (CIDL) – a declarative language used to describe the structure of component implementations. From the information obtained from the CIDL and IDL descriptions of the component, the CIDL compiler generates its implementation skeleton.

Components are packaged to be distributed and deployed. A package contains one or more component implementations and descriptors. Since one component may have implementations targeted at different platforms, one package may contain more than one implementation. The descriptors are written in XML [8] and specify the binary files necessary to deploy components in different platforms, as well as the CORBA services used by components. CCM also establishes rules to deploy a component-based application, enabling the automation of the deployment process and contributing to increase component reuse.

3. Description of the AFT-CCM Model

According to Kim and Lawrance [9], adaptive fault-tolerance is an approach to meet the dynamically and widely changing fault tolerance requirements by efficiently and adaptively utilizing a limited and dynamically changing amount of available redundant processing resources. Large-scale distributed systems, such as the ones built with resources available through the Internet, can benefit from adaptive policies. Adaptive fault-tolerant mechanisms use adaptive policies for managing redundancies, in order to maintain efficiency despite frequent changes in the computing environment.

An adaptive program usually is seen as one that can have its configuration modified due to changes in its system environment. Changes in configuration based on adaptive fault-tolerance approaches can be triggered, for example, by changes in the adopted communication patterns, by the frequency of partial failures in the system or by new necessities of the application [10]. The adaptive fault-tolerance can involve the exchange of algorithms and policies during execution time to adapt the application configuration to changes in the system environment [11].

The proposed model employs adaptive fault-tolerance using the flexibility that arrives from the adoption of a component-based technology. In this project we have chosen to use the CORBA Component Model [2] 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 AFT-CCM model (Adaptive Fault-Tolerance on the CORBA Component Model) allows the application programmer to specify quality of service (QoS) requirements, defining desired levels of dependability of the service. The proposed model introduces logical entities – i.e. components – that provide the non-functional code that monitors the fault occurrences in the application and adapts its configuration in order to maintain the desired level of QoS specified by the user. Reached configurations employ replication techniques aiming to fulfill such requirements.

Figure 2 illustrates the different components that form the execution-time support of the AFT-CCM model in a possible configuration in which components are disposed on four hosts of a distributed system. 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 2, a component identified as *the primary server* is located on host 2; its replica is placed on host 3 (*the backup server*). In our model, the coordination protocol of the replication technique used is implemented by a component: *the replication coordinator* (figure 2). In this way, when the passive replication technique[2] 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 by replacing this component with another coordinator that implements a different replication technique.

The client gets access to the services of the component through a *connector*, which hides from the client eventual changes in the configuration of the application. 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 re-directs 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)

[2] In the passive replication technique [12], after executing a client request, the primary server must send a copy of its state to synchronize backup replicas.

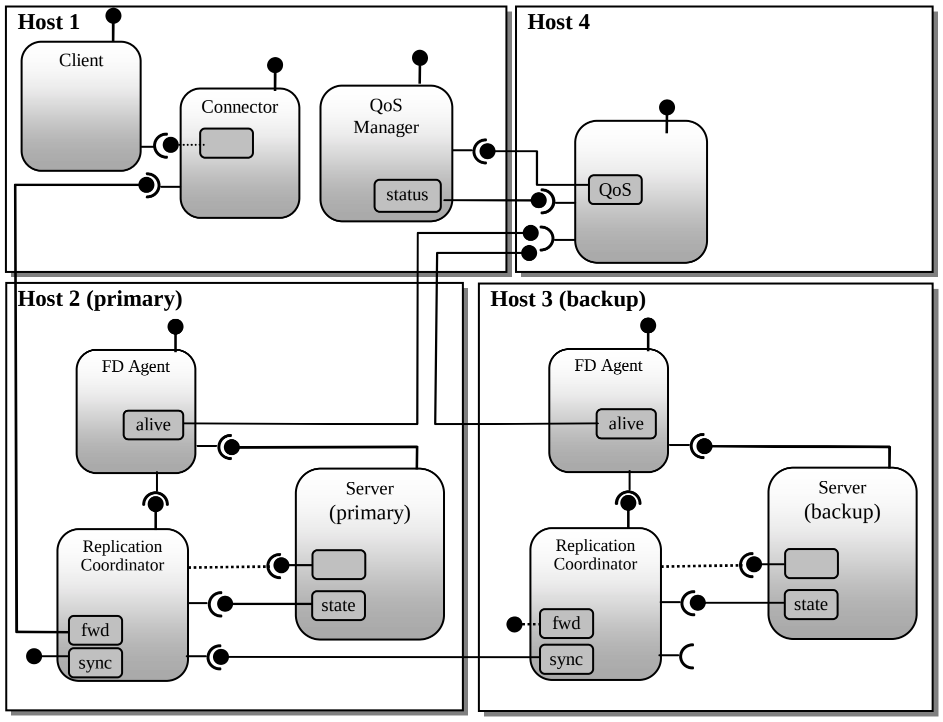


Figure 2. Overview of AFT-CCM

or using a replication technique weaker or stronger than necessary.

Faults are monitored using the *fault detection agents* (FD agents, in Figure 2). Each component replica and replication coordinator is bound to a FD agent, which is responsible for sending monitoring data to the AFT manager. By using the FD agents, two levels of faults can be detected: host faults – in case the FD agent stops responding – and component faults – i.e., when a component stops answering calls from the FD agent.

The AFT-CCM model assumes that hosts and components are subject to crash faults and that the communication between components is reliable. This last premise is based on the fact that the CORBA remote method invocation mechanism provides *at-most-once* semantics and benefits from the error and flow control mechanisms supplied by the IIOP/TCP/IP stack.

In this context, the CORBA exception support is employed to detect component and host crashes. For detecting host faults, the FD agent receives periodically “are you alive?” messages from the AFT manager through CORBA method calls. Host faults are assumed when the call to the FD agent results in an exception. In the absence of faults, this call returns monitoring information to the AFT manager.

For detecting component faults, the FD agent periodically calls the `_non_existent` method provided by class CORBA::Object, which is inherited by all components. Component faults are assumed when the call to the component results in an exception, being reported periodically to the AFT manager by the FD agent.

Both component and host crash faults are handled by the AFT manager, which tries to adapt the configuration of the application, restoring faulty replicas or changing the replication technique, taking into account the QoS requirements specified by the user.

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. As shown by Figure 3, these requirements can be seen as states, and the occurrence or absence of faults can trigger state transitions. The requirements specified by the user 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

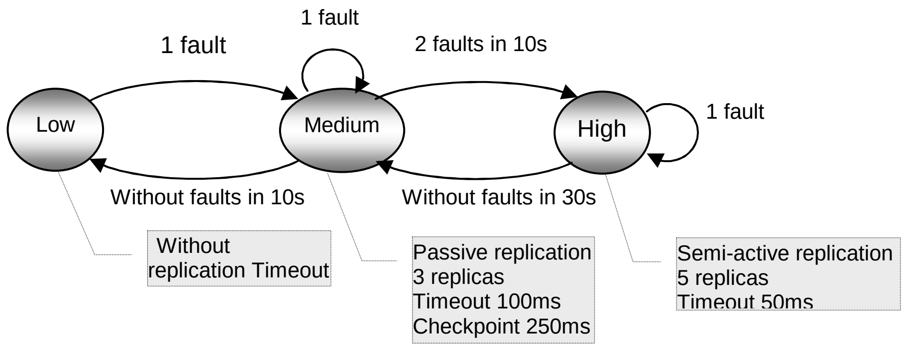


Figure 3. Example of QoS Specification

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.

4. AFT-CCM Implementation

As presented in the previous section, the AFT-CCM model is composed of a set of software components that together aim to provide fault tolerant support for application services in distributed environments. These components were implemented in a prototype using OpenCCM version 0.2 [13], which is a partial implementation of the CCM specification. OpenCCM was chosen because it was the first open source implementation available of CCM. This version of OpenCCM runs on three different ORBs. In this project we have adopted ORBacus version 4.0.5 [14].

4.1. Implementation Issues

Several issues had to be sorted out in order to build the prototype of the AFT-CCM model. These issues and the adopted solutions are described along this section.

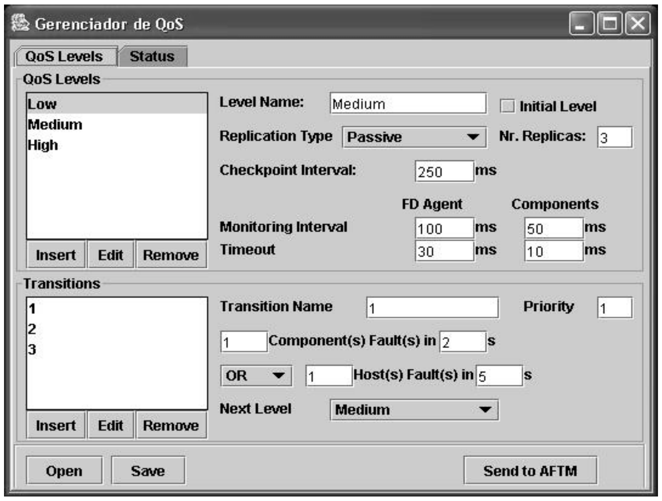
In order to receive invocations for servers with different interfaces, the connector must be generic, i.e. independent of IDL types accepted by the communication ports of the server. However, the establishment of a connection between components implies that the interconnecting ports must have the same type. CORBA's dynamic skeleton interface (DSI) was used to implement the connector, in order to make it generic. This dynamic invocation is represented in Figure 2 with a dashed line. When a invocation arrives at the connector, it simply forwards the invocation to the primary replication coordinator through a connection with its `fwd` facet.

Replication coordinators implement replication techniques. Three different replication coordinators are provided by this prototype: a void coordinator, which does not implement any replication technique, a second that implements the passive replication technique and a third coordinator that implements the semi-active technique (semi-active replication is described in [15]). Active replication was not implemented, because it requires group communication mechanisms that are not provided by ORBacus.

Every replication coordinator has two facets: `sync` and `fwd`. The `sync` facet is used by passive and semi-active coordinators to synchronize state between the primary replica and its replicas. Moreover, the passive coordinator uses this facet to synchronize the state of replicas when the state of the primary component changes. The `fwd` facet is implemented by all replication coordinators to receive client invocations from the connector. Beyond this use, this facet is also used by the leader coordinator in semi-active replication, to forward the invocations received from clients to the coordinators connected to the followers. The `fwd` facet has only a method responsible for calling the method invoked by the client on the replicated component. Since the coordinator does not know previously to which IDL interface (communication port) it is associated, CORBA's interface repository and dynamic invocation interface (DII) are used to discover at execution time how to build and issue a method invocation on a component.

Components must implement the `state` interface (Figure 2) in order to provide a standard way to update the state of their replicas. This interface provides operations to recover (`get_state`) and update state (`set_state`). These operations are identical to the operations for accessing state defined by the FT-CORBA specification [16].

In order to monitor component replicas, the AFT manager calls method `is_alive` from the `alive` facet of the FD agent in time intervals defined by the QoS Manager. As a result of this call, the AFT manager receives information about the situation of the monitored components.


Gerenciador de QoS

QoS Levels | Status

QoS Levels
- Low
- Medium
- High

Insert | Edit | Remove

Level Name: Medium ☐ Initial Level
Replication Type: Passive — Nr. Replicas: 3
Checkpoint Interval: 250 ms

	FD Agent	Components
Monitoring Interval	100 ms	50 ms
Timeout	30 ms	10 ms

Transitions
- 1
- 2
- 3

Insert | Edit | Remove

Transition Name: 1 — Priority: 1
1 Component(s) Fault(s) in 2 s
OR 1 Host(s) Fault(s) in 5 s
Next Level: Medium

Open | Save | Send to AFTM

Figure 4. Graphical User Interface of the QoS Manager

For deploying components, the AFT-CCM uses the deployment APIs specified by CCM and provided by OpenCCM. However, some APIs are not implemented in version 0.2 of OpenCCM, such as the API for configuration of attributes in execution time, making it necessary to implement such functionality.

The AFT manager maintains a data structure with the global vision of all the components deployed in the system. This structure contains references, the location, the replication technique used by each component and other additional information. This data allows the AFT manager to accomplish the necessary reconfigurations in the system, for example, to establish connection between component's ports or to remove components from a host. The AFT manager also uses this data to supply information about system execution to the QoS Manager, such as the replication technique that is being used, the location of replicated components and the failures suffered by the components. The failure of the AFT manager and the consequent loss of this information would compromise the recovery of faults.

To prevent loss of the state information kept by the AFT manager, this data is stored on a persistent storage device. Thus, in case this component presents failure and has to be deployed on another host, its state will be recovered. This becomes necessary because OpenCCM 0.2 does not implement containers, which are responsible for managing the persistent state of components. If the container was available, the AFT manager would have been implemented as an *entity* component, which has persistent state that can be managed transparently by the container. The remaining components that compose AFT-CCM would have been *session* components, because losing the state of these components does not compromise the consistency of the system.

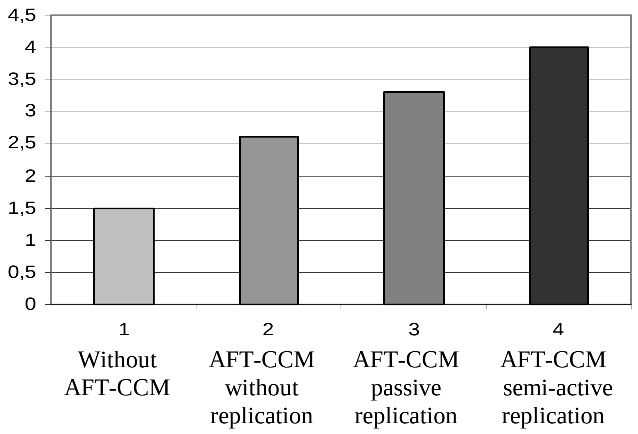


Figure 5. Performance of the AFT-CCM Prototype

Finally, the QoS Manager was implemented with a graphical interface that allows the user to specify QoS requirements (Figure 4). The QoS manager receives status information about the system from the AFT manager through the `status` facet. This information is shown to the user through the graphical interface of the QoS manager. When the QoS Manager notices that it is not receiving information from the AFT manager, it reinstalls the AFT manager in another host.

4.2. Obtained Results

In order to verify the performance of the prototype implementation of the AFT-CCM model, some tests were executed on a testbed composed by Pentium IV 1.6Ghz computers with 256Mb of memory, running Linux Mandrake 9.0 operating system and Java 1.4, linked by an Ethernet 100Mbps local network.

The first test[3] measured the response time of calls to the replicated component in different configurations, that is, with different replication techniques. The following configurations were mounted: one without the addition of the AFT-CCM, with a coordinator who does not implement any replication technique, a second with a passive replication coordinator, and a third configuration with a semi-active replication coordinator, using two replicas in the last two cases. The tests were run on a component with only one facet with an empty method. In this way, the influence of the processing time of the method in the results was minimized, evaluating only the overload generated by the addition of AFT-CCM components.

It is possible to observe on the graph presented in Figure 5 that the AFT-CCM model without replication increases the response time in about 1.1ms in relation to the direct call to the component. This difference reaches

[3] These results were obtained from the average of 1000 executions.

1.8ms using the passive replication technique and increases to 2.5ms when the semi-active replication technique is used. The rise in response time was expected and can be considered acceptable, taking into account that AFT-CCM includes a set of components responsible for providing fault-tolerance requirements and to do so they have to intercept calls and coordinate replicas.

The second experiment[4] verified the average time spent to create a component replica in a new host together with its replication coordinator and its FD agent, and the average time to exchange the replication technique (exchange replication coordinators). It was observed that the average time to deploy a new replica was of 330ms, while the time for alternating between passive and semi-active replication was of 260ms.

5. 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. QuO (Quality Objects) [17] is an example of architecture that gives support for the development of distributed applications based on CORBA with QoS requirements. QuO provides ways to specify, monitor and control the QoS requirements of applications, as well as to adapt its behavior due to changes in the environment. Applications developed with Quo specify their QoS requirements at application level through QoS contracts.

The AQuA architecture (Adaptive Quality of Service Availability) [18] 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 to specify QoS requirements at application level, and the Proteus dependability manager [19] to configure the system in response to faults and availability requirements. Ensemble [20] 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 [21] 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. 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 fault history database.

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 our proposed model strategies can be defined according to the application requirements.

Chamaleon [22] 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.

Some proposals employ dynamic configuration mechanisms in order to tolerate partial failures in a distributed system. Some of these experiences also use the concept of connectors. In [23], a connector is a configuration support that selects from a set of server components the one that has the most adequate signature to execute an invocation. If an exception occurs during the execution, a search for another server component is immediately initiated. The idea of connector presented in our approach is more similar to that introduced in [24]. The connector described in this work concentrates all the non-functional aspects of a replication. In the AFT-CCM model, connectors assume a much simpler role. They are

[4] These results were obtained from the average of 100 executions.

call forwarding elements, which only dispatch calls from clients to the replicated components. Non-functional aspects are treated by the replication coordinators. Our option for a simple connector was based on the fact that CCM already has containers, which are responsible for many of the non-functional aspects of applications.

In [25] 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.

6. Conclusions

This paper presented the AFT-CCM model, which offers flexible mechanisms for building software based on the CORBA Component Model with fault-tolerance requirements. The AFT-CCM model adapts the configuration of applications taking into account the QoS requirements associated with components and the faults that occur in the system.

In order to provide support for QoS requirements, the AFT-CCM was built from a set of non-functional software components, that combined provide fault-tolerance requirements to component-based applications. Although adaptive techniques have been employed in several other works to provide fault-tolerance requirements, these works do not take advantage of the flexibility provided by the use of software components.

A prototype implementation of the AFT-CCM model has been built. Performance measurements have shown that the model adds a small overhead to the application, with the great advantage of providing dependability.

References

[1] Szyperski, C. Component Software: Beyond Object-Oriented Programming. ACM Press/Addison-Wesley Publishing Co, 1998.

[2] OMG. CORBA Components. OMG Document formal/02-06-65, 2002..

[3] OMG. The Common Object Request Broker Architecture v3.0. OMG Document formal/02-06-33, 2002.

[4] Sun Microsystems. Enterprise JavaBeans Specification. v2.0, 2001.

[5] Microsoft. Overview of the .NET Framework. MSDN Library White Paper., 2001.

[6] Bachman, F. et al. Volume II: Technical Concepts of Component Based Software Engineering. Technical Report, Technical Report CMU/SEI-2000-TR-08. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.

[7] OMG CORBAservices: Common Object Services Specification. OMG Document formal/98-12-09, 1998.

[8] W3C. eXtensible Markup Language (XML) v1.0. World Wide Web Consortium, 1998.

[9] Kim, K.H. and Lawrence, T. Adaptive Fault Tolerance: Issues and Approaches. In Proceedings of Second IEEE Workshop on Future Trends of Distributed Computing Systems. p. 38-46, Cairo, Egypt, 1990.

[10] Hiltunen, M. and Schlichting, R. Adaptive Distributed e Fault-Tolerant Systems. International Journal of Computer Systems Science e Engineering, 11(5):125–133, 1996.

[11] Chen, W.-K., Hiltunen, M. A., and Schlichting, R. D. Constructing Adaptive Software in Distributed Systems. In 21st International Conference on Distributed Computing Systems, 2001.

[12] Budhiraja, N. et al. The Primary-Backup Approach. In: Distributed Systems, Chapter 4. Addison-Wesley, 2nd Edition, 1993.

[13] Marvie, R., Merle, P., and Vadet, M. The OpenCCM Plataform., 2002. http://corbaweb.lifl.fr/OpenCCM/

[14] Iona Technologies. ORBacus for C++ and Java, version 4.0.5, 2001.

[15] Powell, D. Delta-4 Architecture Guide. Esprit II P2252, Delta-4 Phase 3, 1991.

[16] OMG. Fault-Tolerant CORBA Specification v.1.0. OMG Document ptc/2000-04-04, 2000.

[17] Zinky, J. A., Bakken, D. E., and Schantz, R. E. Architectural Support for Quality of Service for CORBA Objects. Theory e Practice of Object Systems, 3(1), 1997.

[18] Cukier, M. et al. AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. In 17th IEEE Symposium on Reliable Distributed Systems, 1998.

[19] Sabnis, C. et al. Proteus: A Flexible Infrastructure to Implement Adaptive Fault Tolerance in AQuA. In 7th IFIP Internation Working Conference on Dependable Computing for Critical Applications, 1998.

[20] Hayden, M. G. The Ensemble System. PhD thesis, Cornell University, 1998.

[21] 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.

[22] Bagchi, S. et al. The Chameleon Infrastructure for Adaptive, Software Implemented Fault Tolerance. In 17th Symposium on Reliable Distributed Systems, 1998.

[23] Batista, T. V. and Carvalho, M. G. Component-Based Applications: A Dynamic Reconfiguration Approach with Fault Tolerance Support. Electronic Notes in Theoretical Computer Science, volume 65. Elsevier Science Publishers, 2002.

[24] Sztajnberg, A. Flexibility and Separation of Concerns on the Design and Evolution of Distributed Systems. PhD Thesis (in portuguese), COPPE/UFRJ, 2002.

[25] Marangozova, V. and Hagimont, D. An Infrastructure for CORBA Component Replication. In 1st IFIP/ACM Working Conference on Component Deployment, 2002.