Artigo publicado The 9th IEEE International EDOC Conference (EDOC 2005). 19-23 September 2005, Enschede, The Netherlands

FTWeb: A Fault Tolerant Infrastructure for Web Services

Giuliana Teixeira Santos[1], Lau Cheuk Lung[1], Carlos Montez[2]

[1]Graduate Program in Applied Computer Science- PPGIA
Pontifical Catholic University of Paraná – PUCPR - Curitiba - Brazil
[2]DAS/UFSC – Department of System Automation – Federal University of Santa Catarina – UFSC
Campus Universitário, Caixa Postal 476 – CEP 88040-900 – Florianópolis – SC – Brazil

gtsantos@hsbc.com.br, lau@ppgia.pucpr.br, montez@das.ufsc.br

Abstract

The web services architecture came as answers to the search for interoperability among applications. In recent years there has been a growing interest in deploying on the Internet applications with high availability and reliability requirements. However, the technologies associated with this architecture still do not deliver adequate support to this requirement. The model proposed in this article is located in this context and provides a new layer of software that acts as a proxy between client requests and service delivery by providers. The main objective is to ensure client transparent fault tolerance by means of the active replication technique. This model supports the following faults: value, omission and stops. This paper describes the features and outcomes obtained through the implementation of this model.

Keywords: Web services, Fault tolerance, FT-CORBA

1. Introduction

As the Internet became popular, several application development technologies came up offering dynamic and interactive services, giving rise to e-services, such as: electronic commerce (e-commerce), electronic government (e-gov), among others. However, each technology has its own specific operational environment, which makes it difficult to integrate the different applications.

In order to facilitate this integration, with the intent of defining open standards, groups of specialist companies pooled together creating consortiums and defining SOAP (Simple Object Access Protocol) [SOAP, 2003], WSDL (Web Services Description Language) [WSDL, 2001] and UDDI (Universal Description, Discovery and Integration) [UDDI, 2002]. These sets of protocols and standards, along with other related ones being defined by these consortiums and the Academia, have characterized a new paradigm in application development: the web services.

The key word in web services is interoperability, software components that can be accessed by way of consolidated and widely used protocols like HTTP (Hypertext Transfer Protocol) and XML (Extensible Markup Language)[XML, 2000]. The main advantage lies in allowing the integration of components that have already been developed this flexibility would make it possible to explore the best features of each technology involved in the process of developing a distributed application.

Due to an open architecture, web services have shown themselves to be an excellent option in programming distributed systems, allowing solutions to be developed that suited the heterogeneous and complex nature of these environments. However, in order to fully explore the potential offered by web services it becomes necessary to define a development infrastructure addressing the requirements of reliability and high-level availability. This infrastructure must be flexible enough to be able to preserve all the characteristics of web services.

There still has been little work done in addressing fault tolerance requirements for web services. The main problem faced in proposing a fault tolerant infrastructure in this area is on the fact that web servers do not maintain an active connection throughout all the client's requests and, as a consequence, becoming stateless. For this reason, critical applications built on Internet protocols deploy simplified techniques. For example, they use basic mechanisms that detect the fault and direct future requests to redundant servers. These mechanisms are not capable of tolerating faults while processing a request.

Currently there are no standard specifications dealing with fault tolerance in web services. The propositions found in literature [Aghdaie, Tamir, 2002], [Dialani et. al., 2002], [Deron et. al., 2003] deliver fault tolerance on web services based on the passive replication approach [Budhiraja et. al., 1993] and implement basic fault on primary detection mechanisms and activation of a secondary.

In this paper, we are proposing the FTWeb infrastructure for tolerance of faults on web services. This infrastructure features a set of components and services, some based on OMG's FT-CORBA standard's models and concepts [OMG, 2002], for the development of fault tolerant distributed applications. The FTWeb infrastructure has components responsible for calling concurrently the service replicas, wait for processing, analyze the responses processed, and return them to the client. FT-Web supports the use of the active replication technique in order to obtain fault tolerance in service-oriented architectures. The objective of this approach is to provide tolerance in the following kind of faults: stop, omission and value.

This paper is organized as follows: section 2 presents the conceptual model and the standards comprising web services. Section 3 presents an overview of OMG's FT-CORBA specifications, and section 4 presents features the FTWeb infrastructure. Section 5 brings aspects related to the implementation. The model's performance assessment is shown in section 6. Section 7 discusses the related papers and, lastly, section 8 is the conclusion of the paper.

2. Web Services

Web services [WS-ARCH, 2004] are identified by a URI (*Unique Resource Identifier*), and are described and defined using XML. Figure 1 presents the conceptual model for web services.

The service provider is the entity responsible for the publishing a web service on a service register. Any client using a web service created by a provider is called a service consumer. Usually, consumers survey the register where the provider has published a description of the service. Based on this description, consumers can obtain from the server the binding mechanism, and is then enabled to perform the desired web service. A service register is a central location where providers can publish their web services. Through the central register, consumers can find the services and then, bind them.

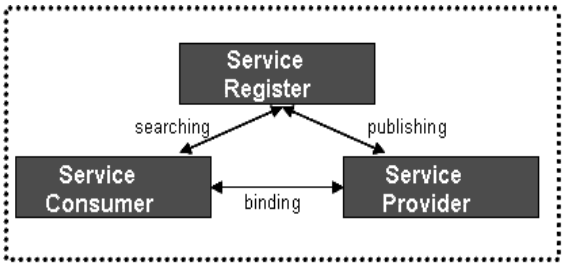

Figure 1. Web Services Conceptual Model.

In order to achieve communication between applications without taking into account the details of their implementation, each operation performed by entities must be standardized. The following standards were created with a view to achieving interoperability: *Web Service Description Language* – a standard that uses XML to describe web services. Basically, the WSDL document defines the methods found in the service, input and output parameters for each one of the methods, data types, transport protocol and the URL for the web service host site. The *Universal Description, Discover, and Integration* standard that allows service providers to publish details about web services they provide on a central register. It also provides a standard to allow consumers to locate providers and obtain details on their web services. The *Simple Object Access Protocol* is an XML-based protocol, used in exchanging information among applications, independent of the operational system, programming language or object model.

3. The FT-CORBA specification

Fault tolerance support for applications developed under the CORBA distributed object model is specified according to the *Fault Tolerant* CORBA (FT-CORBA [OMG, 2002]) standard. This specification defines a set of service interfaces for the implementation of replication techniques in distributed and heterogeneous environments. The fault tolerance architecture in CORBA is shown in Figure 2. The service objects that provide the basic functionalities for building fault tolerant distributed applications are [Fraga et. al., 2001]: *Replication Management Service*, *Fault Management Service and Logging* and *Recovery Service*.

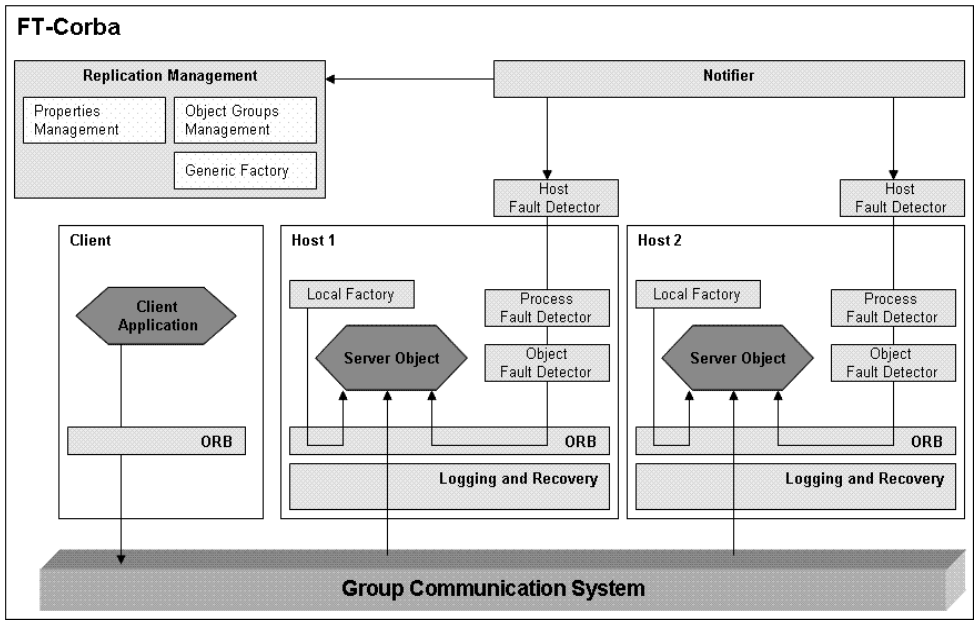

Figure 2. Fault Tolerant CORBA Architecture (FT-CORBA).

The RMS (*Replication Management Service*) interacts directly with the *Object Group Management Service*, acting dynamically in the input and output of replicated objects. In the process of creation and removal of replicas, the object *Generic Factory* is used interacting with the *Local Factory* objects responsible for the creation and removal of replicas at the stations comprising the distributed system. The *Property Management Service* is responsible for defining the fault tolerance properties for each object group. This service defines for the RMS the way in which each group is managed.

The *Fault Management Service* performs the interfaces of the fault monitoring and notification services. Fault detection is carried out in three levels: server, object and process. These detectors are based on *timeout mechanisms*. The *Fault Notification Service* performs the function of informing RMS of the faults recorded by the detectors. Through this notification, RMS keeps a consistent list of group members.

The main objective of the *Recovery and Logging Service* is registering requests received by the server, keep the state of the replicas consistent and carry out recovery procedures on faulty replicas.

4. Description of the FTWeb Infrastructure

The fundamental idea for the FTWeb model is deploying the active replication technique to achieve fault tolerance in service oriented architecture. The replicas for a given service are organized in a group and all fault free replicas receive, execute and reply to the requests submitted by the client. In order to implement the total ordering, necessary for active replication, the sequencer approach was used [Defago, Shiper, 2000]. Applying this model to web services, already implemented and operational, is quite simplified, because it requires only the insertion of replica state monitoring and recovery methods.

This approach allows replication of objects distributed on geographically dispersed servers (in different domains) and delegates their management to the FTWeb infrastructure. This model absorbs the functionalities provided by *FT-CORBA* and supplies components that invoke CORBA objects in the form of web services. Figure 3 presents the components of this model.

A. WSClient Driver

The WSClient Driver component is responsible for detecting faults in the WSDispatcher Engine component and transfer request processing to the WSDispatcher Engine Backup located on an independent server. This component was defined as an interceptor on the SOAP layer and is located on the client. The objective for this approach is to provide fault tolerance transparently to the application's clients, thus preventing the WSDispatcher Engine from becoming a critical fault point.

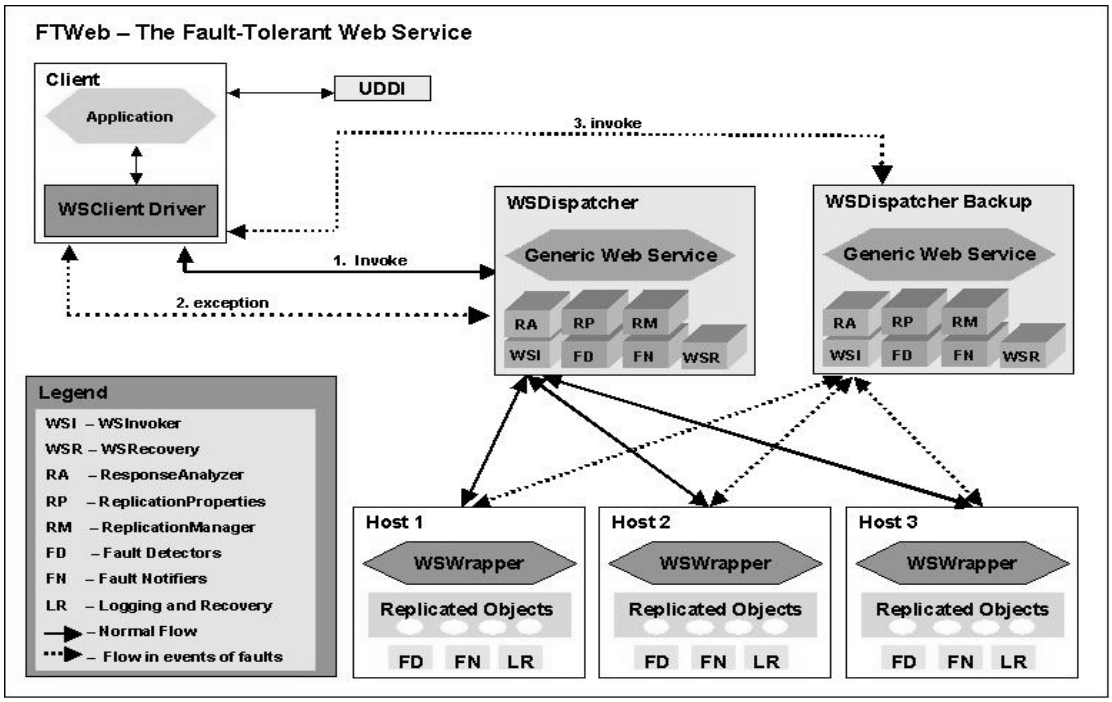

Figure 3. *FTWeb* infrastructure.

In case the WSDispatcher Engine should fail after processing the request, that is, when delivering the answer to the client, the WSClient Driver component will transfer the request to the WSDispatcher Engine Backup that will invoke the replicated services.

Through the log mechanisms contained in the replicas it is possible to check whether the request has already been processed and then, the replicas simply return the answer to the WSDispatcher Engine Backup. Other mechanisms, such as the Finite State Machine [Fred Schneider 1990], can be used to avoid re-processing of requests in case of faults in the intermediate components. Figure 3 shows both, the normal flow and the flow in event of faults on WSDispatcher Engine.

B. WSDispatcher Engine

The *WSDispatcher* is the central component of the FTWeb infrastructure having the mechanisms responsible for replica management, invoking concurrently the service replicas, analyzing the answers processed, detecting and initiating the state recovery process for faulty replicas. The *WSDispatcher* is comprised of the following set of components:

Generic Web Service

The *Generic Web Service* component (Figure 3) is a generic service responsible for obtaining from the client the reference for the web service and the parameters required for its execution. After the execution, this component is in charge of returning the response obtained to the client. Its use makes clients perceive a set of replicated, independent and geographically dispersed services, as a single service.

In order to create the groups, necessary for the replication approaches, the service domain concept was used [Tan et. al., 2004]. A service domain allows aggregation and sharing of multiple web services description (WSDL). The binding information refers to the group, allowing several services to be virtualized as a single service. Rules can be applied to the domain to control the behavior of aggregate services. Figure 4 shows the difference between the service domain model and the conventional web services model (Figure 1).

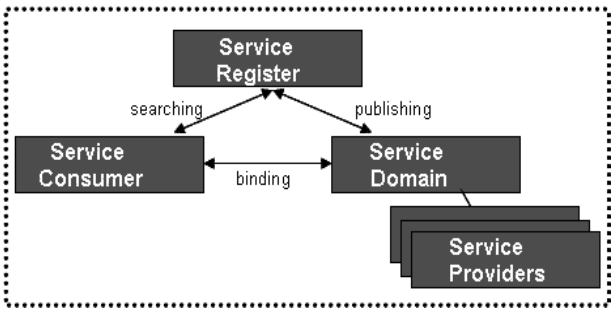

Figure 4. Service Domain.

WSDispatcher Engine contains a Configuration System where the service administrator creates the group and indicates by way of the WSDL documents the replicas that will be part of the group. In this system, the replication and fault management properties are also defined.

WSInvoker

This component is shown on Figure 5. Its operation and integration with the other components can be described by a 5-step sequence:

1. The client invokes the *Generic Web Service* informing the service group reference, the method to be executed and the parameters required to invoke it.
2. The *Generic Web Service* component invokes the *WSInvoker* and passes on the information obtained from the client.
3. *WSInvoker* interacts with the *Replication Manager* and *Replication Properties* components to obtain the location of the replicas and the fault tolerance properties defined for the group.
4. *WSInvoker* invokes the service replicas from the different domains and manages their execution.
5. After obtaining the responses from all the replicas, *WSInvoker* invokes the *Response Analyzer* component that carries out the vote among the responses obtained. *WSInvoker* then returns to the *Generic Web Service* the response indicated by the *Response Analyzer*.

WSInvoker acts as a sequencer and ensures determinism among the replicas by way of atomic ordering. This implies in synchronism in service execution, more than one client cannot execute the same service at the same time. Through the determinism achieved by the *WSDispatcher Engine* model it is possible to use it in *statefull* web services, that is, services that maintain their state information during client requests. Furthermore the WSInvoker provides guaranteed end-to-end delivery of messages based on WS-Reliable Messaging Specification [WS-RM, 2004].

If a replica presents a fault at the moment of its execution or does not respond within the time limit established in the service configuration, the *WSInvoker* component activates the notification mechanisms so that the *ReplicationManager* can remove the faulty replica from the service group. In this case, the faulty replica stays out of the group until its state has been reestablished through the recovery mechanisms.

Response Analyzer

The *Response Analyzer* component is optional (Figure 5) and acts as a voter. After all the replicas have been executed, the *WSInvoker* component delegates to the *Response Analyzer* component the analysis of all responses obtained. The response with the highest number of occurrences is assumed. This component can be used in value fault tolerance.

Replication Manager

The *Replication Manager* component extends the FT-CORBA's replica management functionalities to the web services. This component controls dynamically the adding on of new replicas and the removal of faulty replicas according to the rules defined in the *Replication Properties*.

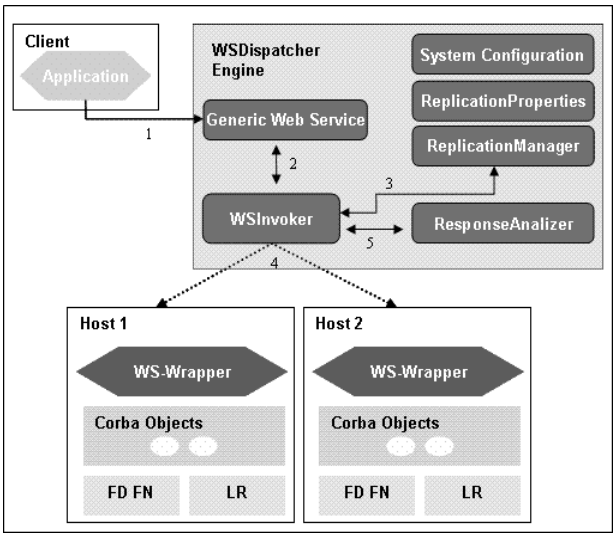

Figure 5. *WSInvoker* operation.

Replication Properties

This component maps out fault tolerance properties defined in *FT-CORBA* for the *FTWeb* infrastructure. As mentioned previously, *WSDispatcher Engine* features a configuration system (shown in Figure 5 as *Configuration System*) that allows the service administrator to define the replication and fault management properties. Through the configuration system, the *Replication Properties* component obtains these properties in XML format. These properties are defined as:

- *Replication Style:* defines the style of replication as cold passive replication, hot passive replication or active replication.
- *Monitoring Style*: defines the style of monitoring between PULL and PUSH. In the PULL style, the

fault detector periodically sends messages to the object being monitored checking whether it is active. In the PUSH style, the object replica periodically sends messages to the fault detector indicating that it is active;

- *Monitoring Interval And Timeout*: defines the monitoring interval (*ping*) and the maximum response time (*timeout*) of the service being monitored in order to determine whether it is faulty;
- *Response Timeout*: defines the response time limit (*timeout*) for the service when invoked by *WSInvoker*;
- *Recovery*: service recovery process indicator. The state of the service can be automatically recovered by means of mechanisms supplied by the *FTWeb* infrastructure or manually by the administrator.

Fault Detector and Fault Notifier

These components extend the fault detection and notification functionalities of the FT-CORBA to the web services. For a web service to be monitored, it is necessary for it to implement the *PullMonitorable* interface containing the *isAlive()* method. By invoking this method, the *Fault Detector* component monitors the replicas. Monitoring is carried out according to the properties obtained through *Replication Properties* and defined in the configuration system.

When a fault occurs, the *Fault Notifier* component receives a fault notification from the *Fault Detector*. *Fault Notifier* notifies *Replication Manager* that removes the faulty replica from the web service group. Figure 6 shows fault management of the *FTWeb* infrastructure.

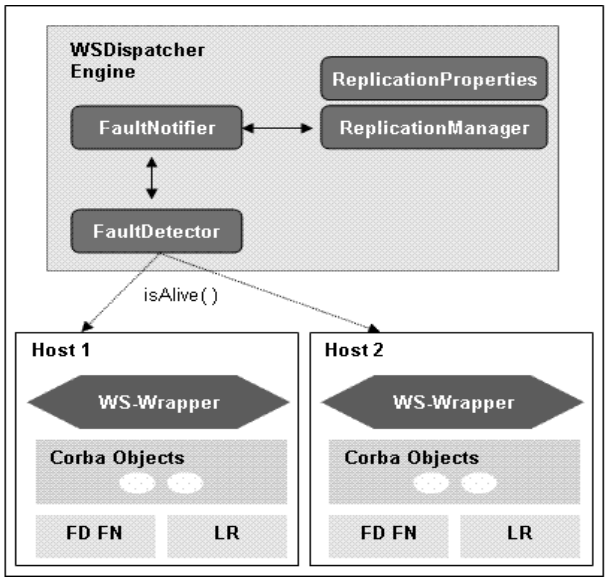

Figure 6. Fault management.

WSRecovery

This component is responsible for the recovery of the state of faulty replicas. The *WSDispatcher Engine* has a monitoring console that displays all the replicas that had faults during the request for a transaction or during the monitoring process. This console allows the service administrator to initiate the recovery process for one or more replicas. The administrator can inform the state of the service in the event of faults in all replicas or initiate the recovery process for just one of the faulty replicas, this process is known as manual recovery.

In automatic recovery, the *WSRecovery* periodically checks the faulty replicas, obtains the state of the non-faulty replicas and through the voting mechanism defined by the *Response Analyzer* component, reestablishes the state of the replica at fault. This component's operation is similar to *WSInvoker*'s (Figure 5), however, its functionality is invoked through the monitoring console or through notification from the *Fault Detector* component when a faulty replica is detected.

C. WSWrapper

In order to explore the integration between the CORBA and web services technologies, a *WSWrapper* component was built to perform the interface between the *WSDispatcher Engine* module and the objects that will process the clients' requests at the provider. This component was based on models defined in [Gokhale et. al 2003] and [Jandl et. al 2003]. Through this component, SOAP requests are converted into CORBA object invocations. *WSWrapper* uses dynamic invocation interface to invoke objects, and can be used in executing any CORBA object on the service provider. Through this approach it is possible to replicate objects on geographically dispersed servers and delegate their administration to *WSDispatcher Engine*.

5. Implementation

The implementation of the *FTWeb* model was carried out using Java JDK 1.4.2, GroupPac 1.4 and JacORB 1.4 languages. The application server used was *IBM WebSphere Application Server 5.1* and all APIs used in this environment are compliant with the interoperability standards defined by *WS-I Basic Profile 1.0* [WS-I, 2004].

The monitoring console and the configuration system were developed in *Java Server Pages 1.2* and *Java Servlet 2.3* allowing administrators to monitor and configure services remotely using only a web browser.

Figures 7 and 8, respectively, show the monitoring console and the configuration system.

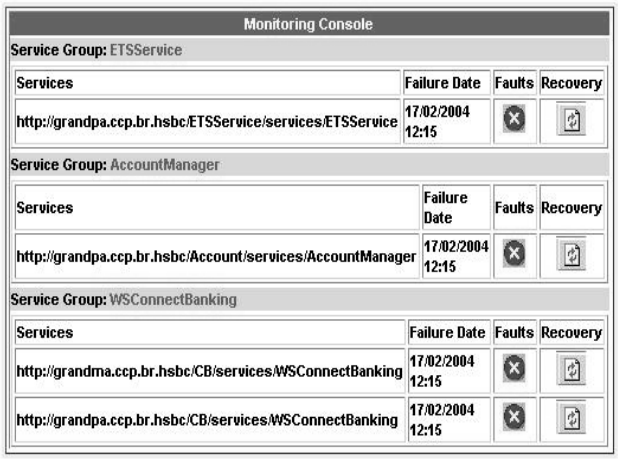

Figure 7. Monitoring console.

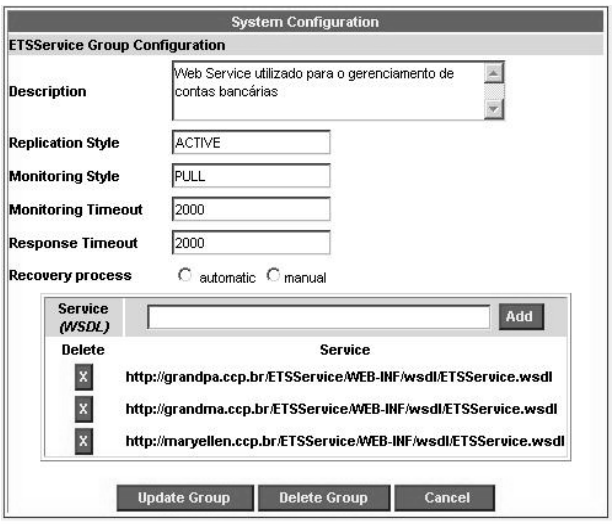

Figure 8. Configuration System.

The *WSInvoker*, *Generic Web Service*, *Response Analyzer* components are made available as a J2EE application on the application server. The *WSInvoker* component executes replicas concurrently using *threads* model provided by the Java language. The *ReplicationManager*, *ReplicationProperties* *WSRecovery*, *FaultDetector* and *FaultNotifier* components are objects implemented under CORBA (Grouppac) fault tolerance specifications, interfaces of objects *ReplicationManager*, *Fault Detector* and *ReplicationProperties* can be viewed in Figure 9.

In order to carry out the monitoring and recovery processes, web services must implement interfaces *PullMonitorable* and *Updateable*. These interfaces can be viewed in Figure 10.

```
public interface ReplicationProperties
{
 void setProperties(ServiceGroup group,
                    Properties props);
 void removeProperties(ServiceGroup group);
 Properties getProperties(ServiceGroup
                                    group);
}

public interface FaultDetector
{
   void registered(Service service);
   void unregistered(Service service);
}

public interface ReplicationManager
{
 void addServiceGroup(ServiceGroup group);
 void removeServiceGroup(ServiceGroup group);
 void addService(ServiceGroup group, Service
                                    service);
 void removeService(ServiceGroup group,
                            Service service);
 void registerNotify(FaultNotifier fault);
}
```

Figure 9. Replication Management Interfaces.

```
public interface Updateable
{
    void setState(State state);
    State getState();
}
public interface PullMonitorable
{
    boolean isAlive():
}
```

Figure 10. Monitoring and Recovery Interfaces.

WSWrapper was developed in order to allow exposure of objects implemented under the CORBA architecture. This component is a web service that performs conversion of SOAP requests into invocations of CORBA objects. Object invocation is performed through the dynamic invocation interface provided by the CORBA architecture. Figure 11 presents the WSDL document of *WSWrapper* service.

FTWeb does not affect the operability of existing services, and can coexist in the same environment, web services managed by *FTWeb* and traditional web services. The deployment of the model on existing operational web services is very simple requiring only the insertion of methods for replica state monitoring and recovery. The model can also be used in different types of web services: *statefull*, *stateless*, synchronous and asynchronous.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://wrapper.corba.services.e
du"
xmlns:impl="http://wrapper.corba.services.edu"
xmlns:intf="http://wrapper.corba.services.edu"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/
soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:types>
<schema elementFormDefault="qualified"
targetNamespace =
"http://wrapper.corba.services.edu"
xmlns = "http://www.w3.org/2001/XMLSchema"
xmlns:impl = "http://wrapper.corba.services.edu"
xmlns : intf = http://wrapper.corba.services.edu
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns : xsd= "http://www.w3.org/2001/XMLSchema">

<element name="execute">
    <complexType>
     <sequence>
      <element name="serviceName"
       nillable="true" type="xsd:string"/>
      <element name="methodName"
       nillable="true" type="xsd:string"/>
      <element maxOccurs="unbounded"
       name="param" type="xsd:byte"/>
     </sequence>
    </complexType>
   </element>
   <element name="executeResponse">
    <complexType>
     <sequence>
      <element name="executeReturn"
       nillable="true"
       type="xsd:anyType"/>
     </sequence>
    </complexType>
   </element>
  </schema>
 </wsdl:types>

<wsdl:message name="executeResponse">
<wsdl:part  element="intf:executeResponse"
   name="parameters"/>
</wsdl:message>
<wsdl:message name="executeRequest">
  <wsdl:part element="intf:execute"
   name="parameters"/>
</wsdl:message>

<wsdl:portType name="WSWrapper">
<wsdl:operation name="execute">
<wsdl:input  message="intf:executeRequest"
   name="executeRequest"/>
<wsdl:output
      message="intf:executeResponse"
      name="executeResponse"/>
</wsdl:operation>
</wsdl:portType>

<wsdl:binding name="WSWrapperSoapBinding"
   type="intf:WSWrapper">

<wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"
/>
<wsdl:operation name="execute">
<wsdlsoap:operation soapAction=""/>
<wsdl:input name="executeRequest">
<wsdlsoap:body use="literal"/>
</wsdl:input>
<wsdl:output name="executeResponse">
<wsdlsoap:body use="literal"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="WSWrapperService">
<wsdl:port
   binding="intf:WSWrapperSoapBinding"
   name="WSWrapper">
<wsdlsoap:address
location="http://localhost:9080/WSWrapper/servic
es/WSWrapper"/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Figure 11. WSDL document of WSWrapper service.

6. Performance evaluation

In order to check the performance of the model proposed, tests were carried out on a 10 Mbps local network composed of Intel Pentium IV 2.8 GHz with 1Gb RAM memory and Microsoft 2000 Professional operational system. *WSDipatcher Engine* was installed on two servers, with one being a backup. Replicas were distributed in up to seven computers, all containing *IBM WebSphere Application Server 5.1*.

In order to assess the overhead of Fault Detector, this component was installed on an independent computer monitoring groups of web services with 3, 5 and 7 replicas. As shown in Figure 12, for groups comprised of up to 7 replicas, CPU use percentage is approximately 4% when the monitoring interval is defined at 2 seconds. Performance appraisal showed that for this execution environment, the best monitoring interval indication for the model is 30 seconds.

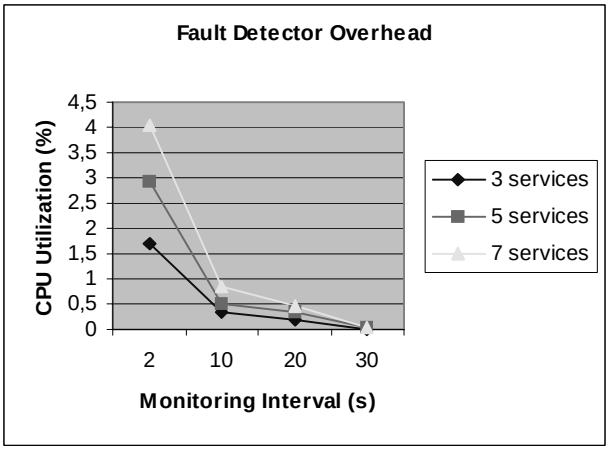

Figure 12. Fault Detector Overhead.

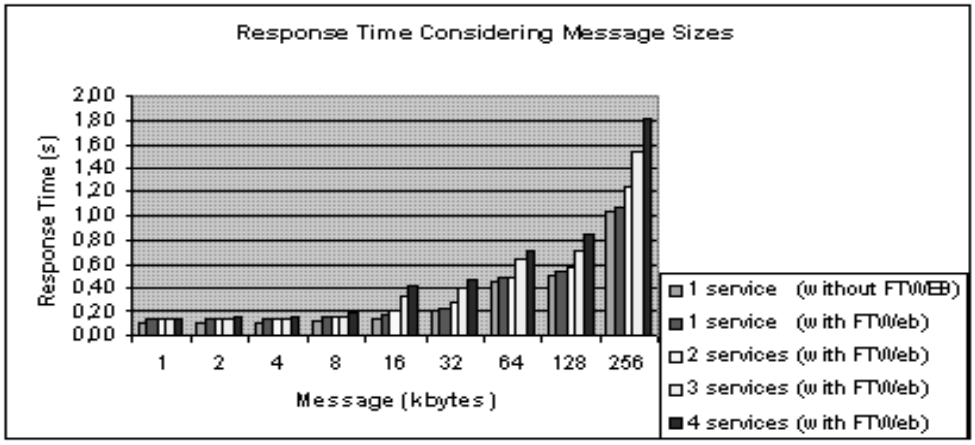

Figure 13. Response Time Considering Message Sizes.

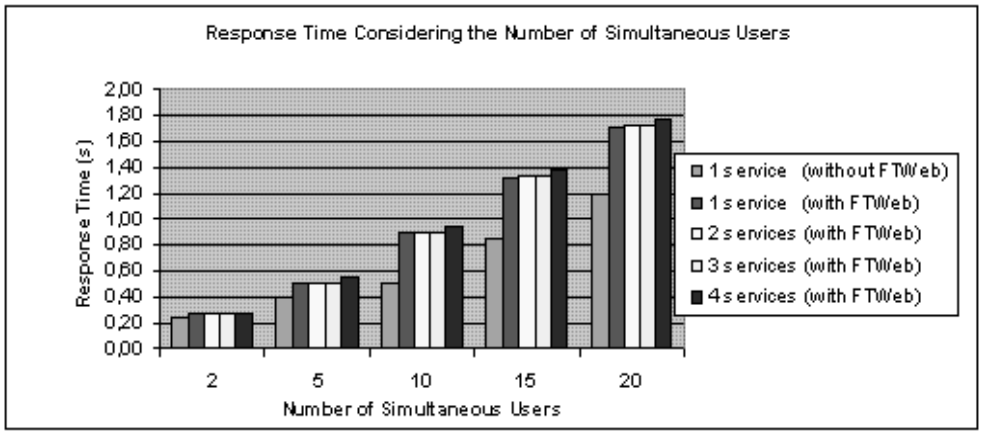

Figure 14. Response Time Considering the Number of Simultaneous Users.

In order to assess the response time added on by the *FTWeb* infrastructure, considering message sizes, tests were carried out with service groups with up to 4 replicas and message size variation between 1 and 256 Kbytes. In order to determine the time added on by the model, tests were carried out for the same services without using *FTWeb*.

It is possible to observe in Figure 13 that for messages with up to 16 Kbytes, variation in the number of replicas comprising the group, does not affect significantly the service response time. Tests presented approximately 25% time added in relation to a service carried out without *FTWeb*. However, the time added to the response time can reach 60%, considering messages with 256 Kbytes and 4 replicas comprising the service using the voting scheme. For tests carried out under the voting scheme, the time was determined by the execution time for the replica located on the slowest machine. Tests executed without voting, sending clients the first message processed, featured equal times, to the tests performed using F*TWeb* with only 1 service in the group.

In order to evaluate the response time added on by *FTWeb*, considering the number of simultaneous users accessing the service, tests were carried out with service groups with up to 4 replicas and variation between 2 and 20 simultaneous users. Message size used was 4 Kbytes. It is possible to observe in Figure 14 that the variation in the number of replicas comprising the group, does not affect significantly service response time.

The response time is affected when the number of simultaneous users is incremented due to mechanisms providing replica determinism. When submitted to 20 simultaneous users, response time added was approximately 41%.

7. Related Works

Even though availability and reliability are vital requirements of critical applications, the works related to propositions for fault tolerant models in service-oriented architecture are quite recent. The model approached in [Deron et. al., 2003] proposes extensions to the SOAP standard allowing deployment of the passive replication technique to achieve fault tolerance. This model carries out alterations on the WSDL document inserting information related to the primary replica and the *backup* replicas. Using interceptors in the SOAP layer at the client allows redirecting of the requests to replicas in case of fault in the primary. On the server, interceptors add on components for *log* records, detection of faults and replica management. *FTWeb* does not perform changes to WSDL documents, services comprising the group are described in the configuration system, following the service domain approach. In the *FTWeb* infrastructure, using interceptors is limited to fault detection in the infrastructure itself, allowing in case of faults on the primary *WSDispatcher Engine* requests can be referred to a *WSDispatcher Engine* backup.

The model proposed in [Aghdaie, Tamir, 2002] carries out changes on the *kernel* of the operating system and the web server providing a fault tolerance mechanism that is transparent to the client. In this model, every request received by the server is registered and sent to a *backup* server. Changes carried out in the kernel of the operating system provide implementation of a *multicast* mechanism allowing requests to be sent to a *backup* server and the primary server. Alterations carried out on the web server allow manipulation and generation of responses to clients. In comparison with this model, FTWeb is more portable, since it acts as merely another software layer not requiring changes in the operational system or the web server.

The work approached in [Dialani et. al., 2002] [Zhang et. al., 2004][Townend, Xu, 2004] proposes fault tolerant models for services implemented and executed under grid service specifications [OGSA, 2003]. In [Dialani et. al., 2002] the main objective of the architecture proposed is detection and recovery in fault situations, this model does not deal with fault tolerance through replication of objects, but rather by means of *checkpoint* and *rollback* mechanisms. In [Zhang et. al., 2004] the passive replication technique is used through notification mechanisms provided by the grid infrastructure. [Townend, Xu, 2004] propose the implementation of a mechanism that carries out a set of equivalent web services, but implemented under different platforms *(n-version).* After the execution of a voting scheme, the model acts on the responses returning the most coincident one.

Despite the theme service grids not being part of the *FTWeb* scope, a few similarities can be found between the models. Similar to service grids, *FTWeb* allows using *statefull* web services and ensures determinism between the replicas. The diversity of programs can be used in *FTWeb*, allowing web services implemented under different architectures to comprise the same service group.

8. Conclusion

This paper presented a proposal for deployment of the active replication technique in order to achieve fault tolerance in service-oriented architectures. This approach provides tolerance for the following faults classes: stop, omission and value. The model proposed is based on a mechanism called *WSDispatcher Engine* comprised of components responsible for: creating service groups, detecting and recovering fault, concurrently invoking service replicas, ensuring determinism among the replicas and establishing voting schemes for the responses returned by the services.

Replicas comprising the service group can be located on geographically dispersed servers, avoiding vulnerability to faults on routers, *gateways* and other network interface components. The application of the model on existing and operating web services is quite simple requiring only the insertion of replica monitoring and recovery methods.

Tests carried out on the prototype show that performance costs are acceptable considering the gains in availability and reliability afforded by the model. Configuration and monitoring of the replicas comprising the service can be performed remotely through a system provided by the *FTWeb* infrastructure, using only a *web* navigator. Aiming future work, there is the implementation of other replication techniques and the integration of the *FTWeb* model with the specifications of service *grids* [WS-RF, 2004].

References

Aghdaie, N., Tamir, Y. (2002). Implementation and Evaluation of Transparent Fault-Tolerant Web Service with Kernel-Level Support. Proceedings of the IEEE International Conference on Computer Communications and Networks Miami, Florida, pp. 63-68

Budhiraja, N., Marzulo, K., Schneider, F. B. e Toueg, S. (1993). Distributed Systems, chapter 4. The Primary-Backup Approach. Addison Wesley. 2nd edition.

Cheuk, L., Padilha, R., Souza, L., Fraga, J. (2001). FT-CORBA Implementation, Technical Report, LCMI-DAS-UFSC, (http:/www.lcmi.ufsc.br/grouppac).

Defago, X., Schiper, A., Urban, P. (2000). Totally Ordered Broadcast and Multicast Algorithms. Technical Report DSC/2000/036, Dept. of Communication Systems, EPFL.

Deron L., Fang, C., Chen, C., Lin, F. (2003). FT-SOAP: A Fault-Tolerante Web Service. Tenth Asia-Pacific Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference, Chiang Mai Thailand pp.310

Dialani, V., Miles, S., Moreau, L. De Roure, D., Luck, M. (2002). Transparent Fault Tolerance for Web Services Based Architectures. Euro-Par 2002. Parallel Processing: 8th International Euro-Par Conference Paderborn, Germany Proceedings. Volume 2400

Gokhale, A., Kumar, B., Sahuguet A. (2002). "CORBAWeb Services", Proceedings of the 11th International World Wide Web Conference, Honolulu, Hawaii.

Jandl, M., Radinger, W., Goeschka, K. M. (2003). Integration of CORBA with Directory Services and Web Services, ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brasil

OGSA Open Grid Services Architecture (2003) www.globus.org/ogsa/

OMG FT-CORBA Specification (2002). Common Object Request Broker Architecture: CoreSpecification Chapter 23. www.omg.org.

OMG WSDL - SOAP to CORBA Interworking (2003) OMG Document. http://www.omg.org

Schneider, F. B. (1990). Implementing Fault-Tolerant Service Using the State Machine Approach: A Tutorial, ACM Computing Survey, 22(4):299-319.

SOAP Simple Object Access Protocol (2003) – World Wide Web Consortium http://www.w3c.org/TR/soap/

Tan, S., Vellanki, V., Xing, J., Topol B., Dudley G. Service Domains (2004). IBM System Journal VOL 43, N4.

Townend, P., Xu, J. (2004). Replication-based Fault Tolerance in a Grid Environment, Proceedings of the UK e-Science All Hands Meeting

UDDI Universal Description, Discovery and Integration (2002) – OASIS http://www.oasis-open.org/committees/uddi-spec/doc/contribs.htm#uddiv1

WSDL Web Services Description Language (2001) – World Wide Web Consortium http://www.w3c.org/TR/wsdl/

WS-ARCH Web Services Architecture (2004) W3C World Wide Web Consortium http://www.w3.org/TR/ws-arch/

WS-I Web Service Interoperability Organization Basic Profile 1.0 (2004). http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

WS-RF OGSI - Open Grid Services Specification (2004). 1.0 http://www-128.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf

WS-RM - Web Services Reliable Messaging Specification (2004). 1.0 ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf

XML Extensible Markup Language (2000) – World Wide Web Consortium http://www.w3.org/TR/2000/REC-xml-20001006

Zhang, X., Zagorodnov, D., Hiltunen, M. (2004). Fault-Tolerant Grid Services Using Primary-Backup: Feasibility and Performance. Cluster 2004, San Diego, California.