Designing Fault Tolerant Web Services Using BPEL

Jim Lau[1], Lau Cheuk Lung[2], Joni da S. Fraga[1], Giuliana Santos [2, 3]

[1]*DAS/CTC – Universidade Federal de Santa Catarina (UFSC), Florianópolis – SC – Brasil*

[2]*PPGIA - Programa de Pós-Graduação em Informática Aplicada*

PUC-PR - Pontifícia Universidade Católica do Paraná, Curitiba – Paraná

[3]*Faculdade de Ciências da Universidade de Lisboa*

Bloco C6, Piso 3, Campo Grande, 1749-016, Lisboa - Portugal

(fraga, jim)@das.ufsc.br, lau@ppgia.pucpr.br, giuliana@lasige.di.fc.ul.pt

Abstract

The web services technology provides an approach for developing distributed applications by using simple and well defined interfaces. Due to the flexibility of this architecture, it is possible to compose business processes integrating services from different domains. This paper presents an approach, which uses the specification of services orchestration, in order to create a fault tolerant model combining active and passive replication technique. This model supports fault of crash and value. The characteristics and the results obtained by implementing this model are described along this paper.

Keywords: *Web services, Fault Tolerance, Orchestration.*

1. Introduction

In the last years, new technologies and standards for software developing has been presented, providing a better integration between applications and services available on the Internet. The web services are part of this scenery and propose a model of distributed services that uses simple accessed and well defined interfaces. Some advantages of this model are:

- Low cost of development: by reusing software components the systems development become faster.
- Integration with legacy systems: it allows the integration with established and operational systems.
- Better interfaces with commercial partners: through electronic interchange of low cost data.

A web service is a software component that accepts requests from other systems through the Internet. The specifications of the web service were created to integrate applications, providing support to synchronous and asynchronous transactions. The web services model is not the first approach of systems' integration, other technologies have the same objective. However, what makes the web services so attractive is the using of open standards, widely used and consolidated protocols.

The XML specification (*Extensible Markup Language*) is used to perform the data exchange. The XML is superficial and extensible and it can easily incorporate business resources as transactions. The web services are built based on existing communication standards that make them independent from transport protocols, being able to be used by HTTP, FTP, SMTP and other protocols.

In order to explore all potential of the web services as an integration model, companies like Microsoft, IBM and BEA have joined efforts to create a standard specification to define and integrate business process. This specification was called *Business Process Execution Language for Web Services* (BPEL4WS) and it defines a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. It extends the web services model and enables the transactions support. The specification defines an interoperable model that makes easier the expansion of the integration of automated processes inside the same corporation and/or between companies (business-to-business). This composition of services is also called orchestration and it includes the business logic and the order of executions, defined from controls flows that cross organizations and applications.

Despite of all the flexibility provided by BPEL and web services architecture, for developing distributed solutions, it is also necessary the standardization of support mechanisms for fault tolerant Web Services that attends the requisites of reliability and availability, fundamental on critical applications. The set of

standardized specifications by W3C [1] and OASIS [2] do not contemplate these requirements and has been motivate some groups of research in the sense of proposing extensions to add mechanisms for fault tolerance in these solutions.

In [3],[4],[5], are proposed fault tolerant models based on the passive replication technique and it implements simplified mechanisms that detect the fault and guide future requests to redundant server. In our previous work [6] we proposed the use of active replication technique with components that converted SOAP requests into CORBA object invocations. In [7] is presented a study using orchestration to implement fault tolerance for web services however in this solution the server that executes the business process logic is a single point of failure. This work represents an initial effort to implement transparently fault tolerance in web services orchestration. Our proposal is an architecture named FTWS-Orch to provide fault tolerance in services-orientated architectures by using a middleware based on services orchestration combining active and passive replication technique.

The paper is organized as follows: a revision of the Web Services platform in the section 2.In the section 3 is introduced the WS-BPEL specification. In the section 4 it is described the FTWS-Orch model. In the section 5 the implementation of the model. In the section 6 the tests and the results are presented. Section 7 the related works are shown and finally in the section 8 the conclusions of the work.

2. Web Services

In the last years, the model of the service oriented architecture became well known as a software architecture that organizes the components of a system in a distributed environment where there are services that can be accessed dynamically through the network.

A Web Service can be defined, in a simple way, as an available and accessible set of operations in global scale through electronic addresses like URL. Another definition to define web services as an interface that describes a collection of operations that are accessible on the network through a mechanism of XML messages.

The Figure 1 presents an usual model of Web Services identifying three types of roles and operations that are performed. The roles presented in the diagram are: service provider, service consumer and register service.

1- Service Provider – responsible for the describing and publishing an specific Web Service in the service register. The provider is also responsible for describing the connection information of the service used for its call. These information are represented in a XML document written in WSDL standard language [8].

2 – Service Consumer – responsible for discovering services, obtaining their description and linking these services to a service provider, in order to invoke the web service, through an URL;

3- Services Register - maintains a directory with information about the services, for instance, name, provider and category. The standard adopted in SOA for registering services is the UDDI [9].

The message exchange between services providers and consumers is done through the SOAP protocol [10]. It is a XML based protocol and it describes the service, and provides all the details needed in order to interact with the service, including the format of the message, the transport protocol and its location

The interaction between the three elements involves: the publication of the information about specific service, the discovery of the available services and the connection between them. The web services architecture presents some advantages some of them could be mentioned: support for different types of clients; the constant need of maintenance; the easy reuse; the scalability which guarantees the needed architecture interoperability, moreover the capacity of sharing and re-using services and resources.

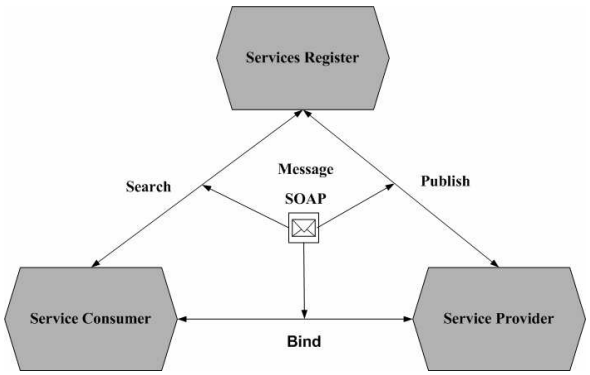

Figure 1. Web Services Conceptual Model.

3. Business Process

The definition of business process involves specifying the behavior of each present participant without revealing its internal execution.

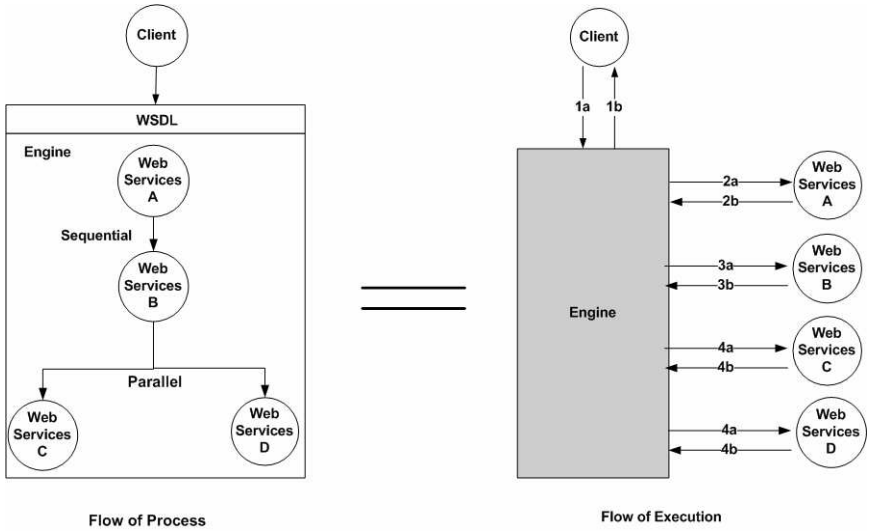

Figure 2. Process Flow and Execution Flow

The separation between public and private aspects allows the independence of modifying the implementation without affecting the business protocol. The basic requirements for describing business protocols are:

- Business protocols invariably depend on behavior data.
- Capability of specifying conditions of exception and their consequences, including recuperation of sequences.
- Capability of working with long business interactions, including multiple occurrences. Each interaction must be seen a work unity with its own requirements. It must have the capability of coordinating various work unities and the concurrent activities in several levels of granularity.

The BPEL4WS defines the grammar to describe the business processes behavior based on the interaction between the processes and his partners. The interaction with each partner takes place through web services interfaces. The process defines how multiple interactions with its partners will be coordinated to perform a specific business process. The BPEL4WS also defines mechanisms to handle exceptions and faults. It also introduces mechanisms to define how an individual activity, or a set of them, can be compensated when exceptions or reverse requests happen from its partner. This services composition is also called orchestration.

The orchestration describes an interaction between services, in the message exchange level, including the business logic and its order of execution. An orchestration is a business process executable, controlled by one of the process members.

Figure 2 presents the execution flow of a BPEL engine. In the left side is shown the business process flow. In the right side is shown the execution flow, controlled by the engine that acts as a "maestro". While receiving a request, the engine starts to compose the orchestration according to the process flow, ordering the execution of each component in sequential or parallel form. The execution flow is the way the engine performs each interaction in its process logic.

The execution flow begins when the client sends a requisition to the engine (1a). In this moment, the first node, or service, to receive this request from the client is the web service A. The engine makes invocation to this service (2a) and it waits for the answer (2b). The next step is the invocation to a web service B (3a), after that it waits for the answer of the service (3b). Next, the engine simultaneously makes a parallel invocation to web services C and D (4a), and the answers of these invocations return to the engine (4b). After executing the process flow an answer, as a result of the execution, is sent to the client (1b).

All invocations in the web services are done from a WSDL document of each service. With that the engine can invoke or receive requests to/from these services. The BPEL offers a standard language that defines:

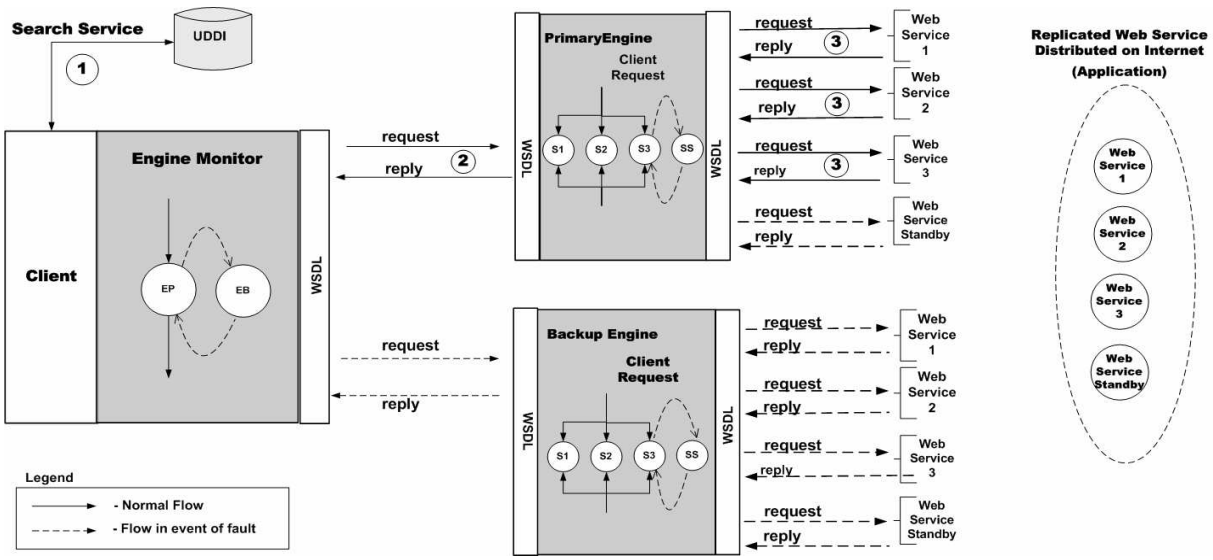

Figure 3. FTWS-Orch Model.

- Sending XML message to a remote service.
- A structure to manipulate the data XML.
- Receiving asynchronous XML messages to remote services.
- Events and the exceptions handling.
- Execution of parallel events.

These are the main items to compose a set of services inside a collaborative and transactional business process. BPEL is based on XML Schema, SOAP and WSDL.

4. The FTWS-Orch Model

The main idea for the FTWS-Orch model is use of the business process specifications in order to define an architecture for development of fault tolerant applications using a combination of active and passive replication.

The replicas of a given service are grouped through the definition of a business process. All replicas implement the same service and receive, execute and reply the requests send by the clients. This model allows the use of synchronous, asynchronous web services and services implemented on heterogeneous platforms or languages (n-version approach).

Through the model specified in this work it is possible to compose business processes using fault tolerant services. The components of this solution can be divided in three principal modules according to their functionalities: Composition and invocation of services, Failure detection, and Fault tolerance transparently to the application's client.

4.1 Composition and Invocation of Services

In order to create the groups, necessary for the replication approaches, a tool of business process management is used and the administrator defines the flow of execution informing the WSDL documents that will make part of the service composition. These documents refer the replicas of a given service. The flow defines that these replicas will be executed concurrently according to active replication technique.

In the client view, this composition works in the same way that a single service, however, these services are replicated, independents and could be located in different domains.

In the Figure 3, after the business process definition, the administrator publishes it in a UDDI register as a traditional web service.

The client looks in the services register and obtains the WSDL document of the business process and it carries out the service invocation (step 1 of the figure 3).

The engine is responsible for the interaction between the client and the replicated web services (step 2 and 3 of the figure 3). The primary engine obtains from the client the reference of the web services composition, the necessary parameters for his execution, manages the execution in all replicas and returns a response to the client. In this model a response is returned to the client when there is at least a fault free replica.

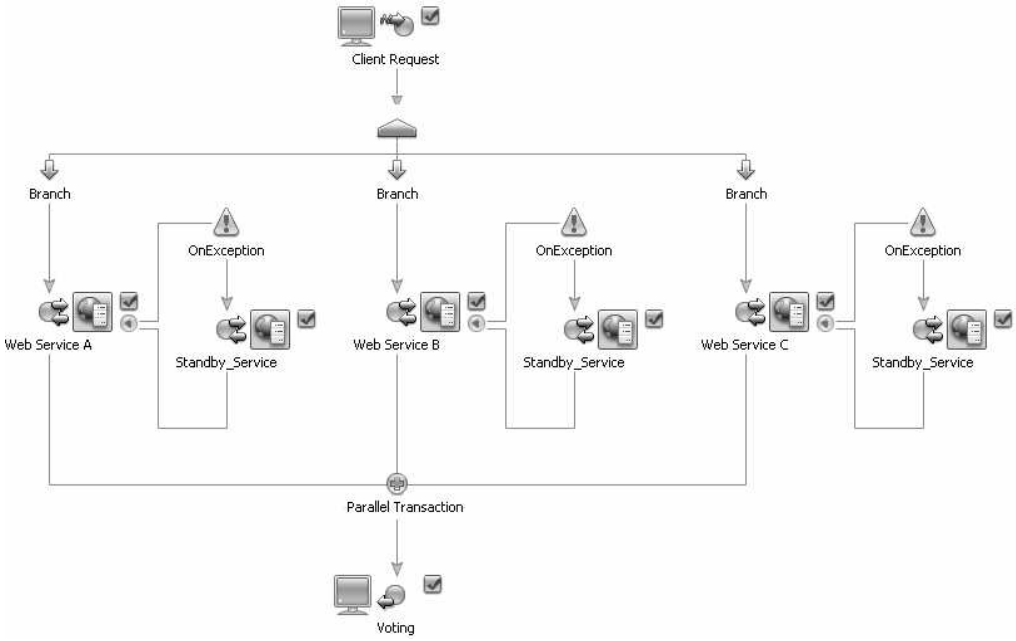

Figure 4. Business Process Composition Interface

4.2 Failure Detection for Replicated Web Service

Due to the monitoring process interfere in the solution performance, here, we do not use monitor components to detect faulty replicas. On the other side, when a replica fails FTWS-Orch transfer the client requests dynamically to a standby replica such as the passive replication technique.

This proposal only work with *stateless* web services therefore, we do not implement recovery mechanisms of replicas state or mechanism to guarantee the service determinism. In a future work these mechanism will be added follow the same approach used in [6].

The flexibility presented in the business process specification allows defining to each replica a specific *standby* replica (SS in the figure 3) or define for all replicas exactly the same *standby*. According to the Figure 3 inside of each Engine (Primary and Backup) has a service composition with a defined *standby* replica.

4.3 Fault Tolerance Transparently to the Application's Client

The facilities of business process are used also in the client side to avoid that the Primary Engine be a critical point of failure. The Primary Engine invocation could be achieved as a business process and an exception in the primary engine is dynamically transferred to Engine Backup using the passive replication technique. All this operation of redirecting is carried out by the engine monitor.

Figure 3 presents the normal flow and the flow in event of failures on Engine Primary. Through a log mechanism contained in the service replicas it is possible to check whether the request was already processed and then the replicas simply return the response to the Engine Backup and the request is not re-executed.

4.3 Tolerating Values Faults

In order to tolerate values fault is possible add in the composition a web service that acts as a voter and choose the response with the highest number of occurrences. When this component is used it is necessary at least 2f + 1 replica to decide and to return the response to the client, where f is the number of faulty replicas, otherwise an error message is returned.

5. Implementation

In order to validate our proposal we implemented a prototype using the *BEA Weblogic Workshop 8.1*[1] tools. A business process was defined varying the number of services. These services were installed in different

[1] http://www.bea.com/framework.jsp?CNT=index.htmFP=/content/products/Weblogic/workshop

computers in a local network. They implemented access to the server file system where the web service is hosted.

The Figure 4 presents the interface of business process composition tool. Services group is represented by Web Service A, Web Service B, Web Service C that are executed concurrently. Differently of other approaches in our work is not necessary to change web services already implemented to include methods for monitoring services. Exceptions on these services are detected and the request is transferred to Standby_Service according to specified in *on exception* flow.

Through this approach is possible to implement fault tolerant business process using a composition as a part of another composition. The model proposed here allow the administrator to change service groups easily. The administrator performs changes in service groups obtaining the WSDL documents of each service and changing the business process to work with new services.

After the complete business process definition is obtained the document XML that will be used by the engine BPEL for the flow execution Figure 5 presents the main parts of the BPEL archive defined in the FTWS-Orch model. Replicas are executed concurrently using the element *flow*. The detection of faults in the primary replicas and the redirecting for *standby* replicas is carried out through the element *fault handlers*.

```
<flow name="Parallel-Transaction"
jpd:name="Parallel Transaction">
  <sequence name="Branch">
    <faultHandlers jpd:name="OnException">
      <catchAll>
        <scope name="Standby_Service">
            …
     <invoke
     name="Standby_Service"
     partnerLink="Standby_Service"
     portType="ctrl:FTWS_Orch.Standby_Servic
     ePT"
     operation="nome"
     inputVariable="input"
     outputVariable="output"/>
     <assign></assign>…
     </catchAll>
      </faultHandlers>
     <scope name="Web_Service_1">
     …
     </scope></sequence>
  <sequence name="Branch">
    <faultHandlers jpd:name="OnException">
      <catchAll>
        <scope name="Standby_Service">
            …
     <invoke name="Standby_Service"
     partnerLink="Standby_Service"
     portType="ctrl:FTWS_Orch.Standby_Servic
     ePT"
     operation="nome"
     inputVariable="input"
     outputVariable="output"/>
     <assign></assign>…
     </catchAll>
      </faultHandlers>
      <scope name="Web_Service_2">
     …
     </scope>
     </sequence>
</flow>
```

Figure 5. FTWS-Orch BPEL.

6. Performance Evaluation

In order to check the performance of the proposed architecture tests were executed in a 100Mbps local network composed of Intel Pentium 4 2.0GHz with 1Gb RAM and Windows XP operational system. The prototype was created in the BEA Weblogic Workshop tool. The engine FTWS-Orch was installed in two server, one being backup server. The replicas were distributed in up to four computers all containing Web Apache Tomcat version 5.5.12 integrated with the server Web Service (Axis).

In order to check the response time added by the FTWS-Orch considering message sizes, tests were carried out with variation in the messages size from 1 to 32 Kbytes. The limitation in the messages size is due to platform used that does not allow that messages bigger than 32 Kbytes be used in the business process composition.

It is possible to observe in Figure 6 that for message with up to 4 Kbytes the variation of the number of replicas comprising the group, does not affect significantly the service response time. Tests presented approximately 23% time added in relation to a service carried out without FTWS-Orch

However, the response time added using the voting scheme can reach to 90 %, considering message with 32 Kbytes and 4 replicas composing the service group. The response time using the voting mechanism is determined by the response time of the lowest replica.

In order to evaluate the response time considering the number of simultaneous users, tests were executed with up to 4 replicas and variation between 2 to 20 users. The message size used to carry out this test was 1 Kbyte. In the Figure 7 it is possible to observe the response time can reach to 14 seconds with 20 simultaneous users.

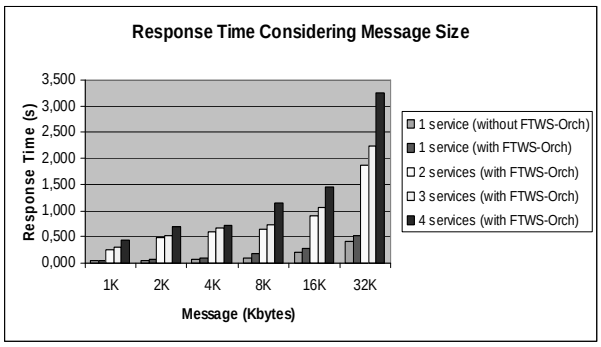

Figure 6. Response Time Considering Message Size

Tests were carried out in order to check the difference in the total response time of a given service when a replica presents a fault and the request is redirected to a standby replica. The average time observed was 1.2 seconds considering a service group with 4 replicas and 1 faulty replica. The response time added for redirecting was 40 % in relation to the time observed with all operational replicas.

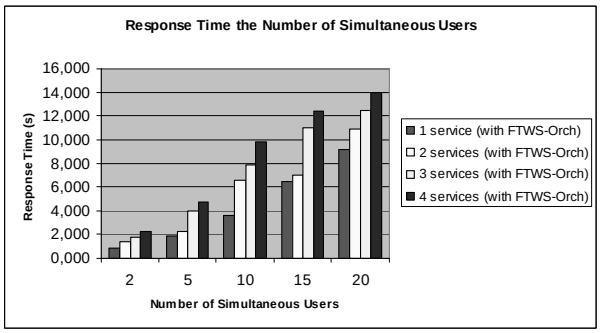

Figure 7. Response Time Considering Simultaneous Users

The response times presented above are bigger than the expected one mainly when compared with our previous work [6]. However in our future work we intend to optimize the BPEL engine in order to reach better results.

7. Related Work

The fault tolerance in service oriented architecture is the key to their widespread uptake in mission-critical applications. There are standards to deal with dependability at the message-passing level and clustering mechanism to implement fault tolerance at the application server. However little emphasis has been placed upon fault tolerance of the web services implementation.

The model approached in [5] 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. FTWS-Orch does not perform changes to WSDL documents and works with elements defined in BPEL. Services groups are created following the services composition approach.

The model proposed in [3] 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, FTWS-Orch 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 [4] [11] [12] proposes fault tolerant models for services implemented and executed under grid service specifications [13]. In [4] 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 [11] the passive replication technique is used through notification mechanisms provided by the grid infrastructure. In [4] proposes 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 FTWS-Orch scope, a few similarities can be found between the models. The diversity of programs can be used allowing web

services implemented under different architectures to comprise the same service group.

The work presented in [7] explores the WS-BPEL in order to achieve fault tolerance in service oriented architectures. In our opinion the main weakness of this approach is not show how faults in the application server, where the business process is executed, can be tolerated therefore in this solution BPEL engine is single point of failure. This work mentioned *stateful* web services but does not present how reach the determinism necessary in the active replication technique.

In our work the flexibility of BPEL standard is extended to implement a business process in the client side allowing in case of faults on the primary FTWS-Orch requests can be referred to a Engine backup. In our previous work the infrastructure FTWEB [6] executes replicas concurrently using *threads* model and has mechanisms to detect faulty replicas. FTWS-Orch manages the execution of replicas using the BPEL engine and combines the use of active and passive replication in order to detect and substitute faulty replicas.

The WS-FTM (*Web Service-Fault Tolerance Mechanism)* [14] is based on [4] and applies N-version technique to the domain of Web Services to achieve increases in system dependability. Differently of our approach all components in this infrastructure are located in the client side.

8. Conclusion

This paper explores the flexibility provided by business process specification and combines the use of active and passive replication technique in order to achieve fault tolerance in service oriented architectures. This approach works with elements of business process specification and provides tolerance for crash and value faults. Our work does not implement monitor system in contrast when a faulty replica is detected requests automatically are referred to *standby* replicas. Through this approach administrators can choose what services are more critical and define only replicas for these services or choose one standby replica for the entire group, these strategies allow reduce the system cost. In other solutions the application server, where are located the web services, represent a single point of failure. We solved this problem extend our model to the client side doing the FTWS-Orch invocation as a service composition.

Tests carried out on the prototype, without voting mechanism, show that performance costs are acceptable considering the gains in availability and reliability afforded by the model. However this work is at an early stage and in the future we intend provide a complete model including *statefull* web services.

References

[1] W3C, "World Wide Web Consortiom," www.w3c.org, 2005.

[2] OASIS, "Organization fot the Advancement of Structured Information Standards," www.oasis.open.org, 2005.

[3] N. Aghdaie and Y. Tamir, "Implementation and Evaluation of Transparent Fault-Tolerant Web Service with Kernel-Level Support," presented at Proc. IEEE Intl. Conf. on Computer Communications and Networks, 2002.

[4] V. Dialani, S. Miles, L. Moreau, D. D. Roure, and M. Luck, "Transparent Fault Tolerance for Web Services based Architrectures " presented at Eighth International Europar Conference (EURO-PAR'02), 2002.

[5] D. Liang, C.-L. Fang, C. Chen, and F. Lin, "*Fault tolerant web service*," *Jornal of Systems Architecture*, 2006.

[6] G. T. Santos, L. C. Lung, and C. Montez, "FTWeb: A Fault Tolerant Infrastructure for Web Services," presented at Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05), 2005.

[7] G. Dobson, "Using WS-BPEL to Implement Software Fault Tolerance for Web Services," presented at Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'06), 2006.

[8] WSDL: http://www.w3.org/TR/wsdl 2001.

[9] UDDI, "Universal Description, Discovery and Integration," http://www.uddi.org/specification.html, 2001.

[10] SOAP, "Simple Object Access Protocol," in *W3C World Wide Web Consortium.*: www.w3c.org/TR/soap, 2003.

[11] X. Zhang, D. Zagorodnov, and M. Hiltunen, "Fault-Tolerant Grid Services Using Primary-Backup: Feasibility and Performance," presented at Cluster, San Diego, California, 2004.

[12] P. Townend and J. Xu, "Fault Tolerance within a Grid Environment," presented at Engineering and Physical Sciences Research Council (EPSRC´05), 2005.

[13] OGSA, "Open Grid Services Architecture," www.globus.org/ogsa, 2003.

[14] N. Looker and M. Munro, "*WS-FTM: A Fault Tolerance Mechanism for Web Services*," Technical Reports, 2005.