# PROVIDING DEPENDABILITY FOR WEB SERVICES[*]

Jeferson L. R. Souza
Distributed Systems Research Laboratory (LaPeSD)
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis - SC - Brazil
jeferson@inf.ufsc.br

Frank Siqueira
Distributed Systems Research Laboratory (LaPeSD)
Department of Informatics and Statistics (INE)
Federal University of Santa Catarina (UFSC)
Florianópolis - SC - Brazil
frank@inf.ufsc.br

## ABSTRACT
Web services have been widely employed to allow interoperability among applications and/or technologies. However, the standard technologies and protocols which provide the foundation for Web Services do not address issues such as fault tolerance and dependability of services. Aiming to solve this limitation, this paper proposes a software architecture for providing dependability for Web Services. This architecture is responsible for increasing service availability and maintaining all replicas of a service in a consistent state, having as main characteristic the separation of these replicas in groups.

## Categories and Subject Descriptors
C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed applications*; D.4.5 [**Operational Systems**]: Reliability—*Fault-Tolerance*

## General Terms
Reliability

## Keywords
Dependability, Fault Tolerance, Web Services

## 1. INTRODUCTION
The Web Services technology has as its main purpose the provision of interoperability for applications built upon it. This interoperability is employed mainly for data exchange, allowing the use of Web Services for system integration thorough the adoption of a standard communication protocol and a data representation format. The adoption of widely available standards allows these systems to exchange information in a uniform fashion, despite being implemented using a wide range of languages and technologies on top of different software and hardware platforms. For some specific systems, such as ERP (Enterprise Resource Planning), the interaction and communication with other systems is very important. Therefore, the failure of a web service required by these systems in order to work properly becomes highly critical. This failure may, for example, interrupt a production line for some hours due to a failure in the integration with the system of a business partner that provides parts necessary for production. Improving the availability of these web services is highly necessary for the adoption of this technology in critical environments.

The more traditional solution for providing fault tolerance in computing systems is the use of replication schemas, which are able to improve the dependability of services [11, 7]. Based on this premise, this paper proposes a fault tolerance architecture for Web Services called WS-FTA (Web Services Fault Tolerance Architecture). This architecture describes a conceptual model in which a set of components, which may be implemented using different technologies, provides fault tolerance mechanisms for Web Services. The proposed solution is based on the current standards adopted web services - i.e., it is fully interoperable with legacy systems.

This paper is organized as follows: section 2 presents a general view of dependability in Web Services. Section 3 presents the WS-FTA model, and its reference implementation is described in section 4. Section 5 evaluates the performance of the reference implementation, and finally, section 6 presents some conclusions and suggestions for further development of this work.

## 2. DEPENDABILITY IN WEB SERVICES
Web Services may be defined as any service, available on the Internet, which communicates using a XML-based standard protocol and is not limited to any operating system and/or programming language. A set of protocols and standards has been adopted for service discovery, interface description and message exchange among web services. These protocols and standards are, respectively: UDDI (Universal Discovery, Description and Integration), WSDL (Web Services Description Language) and SOAP [1].

The Web Services conceptual model has three main entities derived from the Service Oriented Architecture (SOA) [1]. The Service Register maintains information regarding service description and localization. Service Consumers utilize this register to locate the service and to obtain the description of its interface, which is necessary in order to have access to the service. The Service Provider registers itself in a service register, so that service consumers can locate it

[*]This work is partially supported by CNPq, Brazil.

and avail of its services. Service register, service provider and service consumer use the protocols cited previously to interact with each other.

Web Services provide a very interesting technology for conceiving and building distributed systems [14]. One of the main advantages provided by the standard protocols which form the Web Services technology is the interoperability among applications. However, such as any software, Web Services may be affected by programming failures, and since they are executed on top of a platform consisting of an application server, operating system, computer hardware and network infrastructure, they are affected by any failure suffered by this platform. These problems, which may result in service unavailability, can be classified in three types: problems related to the service, problems with the execution platform (hardware and/or software) and network communication problems. Therefore, fault tolerant techniques must be applied with the goal of reducing the impact of service failures.

According to several literature sources [8, 4, 9, 7, 5], fault tolerance in Web Services can be defined as the capacity of providing the requested service in the presence of faults. Fault tolerance techniques can be used to improve Web Services dependability. The provision of fault tolerance is based on the use of techniques that allow a fault to be handled in a way that it does not interrupt the execution of the software. In this scenario, techniques such as state replication [6], failure detection [2], event ordering [11, 10] and reliable communication [3] are employed for providing fault tolerance in the required level.

Considering that a common way to provide dependability for Web Services has not been standardized, the next subsection presents some software architectures and infrastructures that have been proposed in the literature.

### 2.1 Related Work

Ye and Shen [14] have developed a middleware to support the conception of reliable Web Services using the active replication technique. This middleware requires that service consumers are implemented using a set of classes that allow these consumers to access the replicated service with fault tolerance properties. Only operations that change the state of replicas are executed with total ordering. The service administrator must provide a list describing these operations.

Li et al. [7] have defined a framework called SWS, which supports the development of fault tolerant Web Services. This framework employs replication schemas and N-Modular redundancy as conceptual base [13]. The main goal of this work is to use Web Services in critical applications, modeling security attacks as byzantine faults. This work separates replicas in different groups, having an inter-group communication protocol (SWS-IGC) and a message ordering mechanism (SWS-MO) in order to guarantee correct and efficient group communication. Changes in the UDDI registry were necessary for maintaining group membership information. Service consumers must be developed using a specific package provided by the framework.

Santos et al. [9] have proposed a fault tolerant infrastructure for Web Services called FTWEB. This infrastructure is based on the FT-CORBA specification [FT-CORBA2002]. Service replicas are organized in only one group, and all requests issued by service consumers are forwarded to a component called WSDispacher. The WSDispatcher is responsible for managing the group of replicas, receiving the requests and forwarding these requests to all replicas of the service provider. In this work, services are implemented as CORBA objects, using a component called WSWrapper, which is responsible for integrating these objects with the rest of the infrastructure. Service consumers are implemented with the WSDriver component, which allows consumers to avail of the fault tolerant services provided by FTWEB.

Fang et al. [4] have defined a fault tolerant architecture called FT-SOAP. This infrastructure is also based on FT-CORBA specification, such as FTWEB. FT-SOAP employs passive replication to provide service reliability. FT-SOAP is composed by four kinds of components: the replication manager; fault detectors; fault notifiers; and a recovery and logging mechanism. FT-SOAP adds a new element to the WSDL file which stores references to Web Services that integrate a group of replicas of a fault tolerance service.

Salas, J. et al. [8] specify a framework for high availability in Web Services called WS-Replication. The main objective of this framework is the replication of Web Services, organizing one group of replicas using active replication. WS-Replication defines two main components: a replication component and a reliable multicast component. The replication component is responsible for all characteristics involved in active replication and uses the reliable multicast component to send requests to all replicas. This reliable multicast component is based on the SOAP protocol.

Although they allow improving the availability of Web Services, these related proposals present limitations, such as lack of support for legacy consumers (i.e., consumers developed without any fault tolerance support) and incompatibility or the necessity of changes in the existing standards. The architecture proposed in this paper aims to reach similar goals without presenting these limitations.

## 3. THE WS-FTA ARCHITECTURE

The specification of this software architecture provides an abstract description of the required components for the provision of dependability for Web Services. This software architecture is called WS-FTA (Web Services Fault Tolerance Architecture). WS-FTA provides implicit separation of concerns, dividing responsibilities and functionalities among the defined components. These components are:

- Group Communication component (GC): responsible for encapsulating the access to the transport protocol employed to send and receive messages;
- Message Orderer component (MO): responsible for guaranteeing the total ordering of messages in all replicas;
- Recovery component (REC): responsible for recovering the state of a faulty replica;
- Log component (LOG): responsible for registering interactions among service consumers, service providers and WS-FTA components;
- Replica Failure Detector (RFD): responsible for detecting the failure of a single replica;
- Group Failure Detector (GFD): responsible for detecting the failure of a group of service replicas.

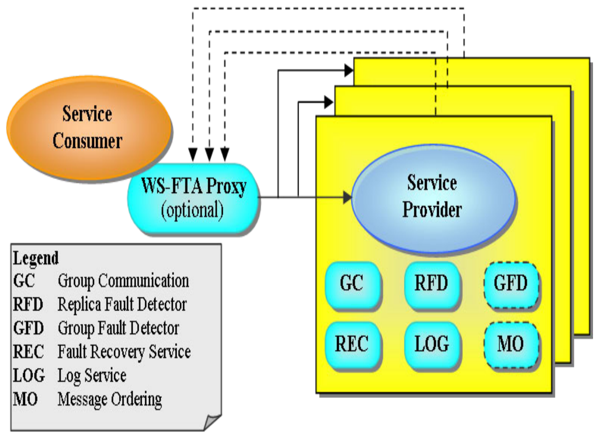

**Figure 1: Components of WS-FTA**

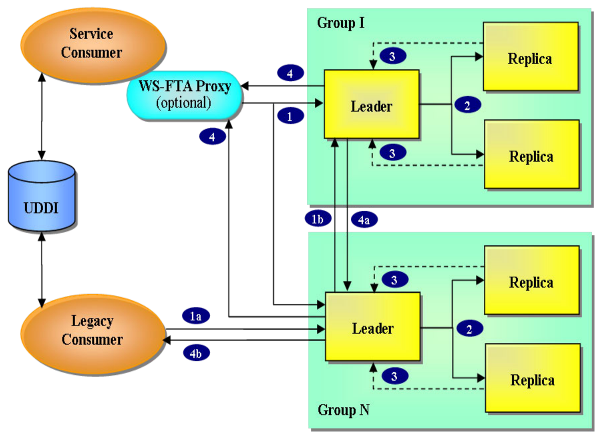

**Figure 2: Replicas organized in two groups**

The architecture proposed in this paper adopts the use of replication as the main strategy to provide dependability for Web Services. Another important design strategy is the organization of replicas in multiple groups. This strategy allows the use of different replication techniques in different groups. Consumers are unaware of replication, and perceive groups of replicas as a single entity.

The architecture allows a minimal configuration in which there is only one group, and all replicas belong to this group. In this scenario, the WS-FTA Proxy will send the request to all replicas and will be responsible for executing the voting algorithm which will select the result that will be sent to the service consumer.

Legacy consumers - i.e., consumers which do not have a WS-FTA Proxy - send a request to just one service replica, which is responsible for forwarding the request to the other replicas. This replica is called the group leader. In this case, replication transparency is provided by the group leader. Depending on the adopted replication technique, the leader may have to collect the results obtained by each replica, and execute a voting algorithm in order to choose the result that will be returned to the legacy consumer.

Figure 1 shows that each host with a service replica runs also a set of components provided by the architecture. Not all of them, however, must be active in all replicas. The MO and GFD components are enabled only in the group leader. The MO component is responsible for defining the order in which received requests will be processed. The GFD, on the other hand, is enabled only in configurations with multiple groups. This component is responsible for detecting the failure of other groups of replicas; therefore, it is needed only when more than one group exists. The creation of multiple groups with replicas of the same service results in a higher level of failure transparency. In a scenario with multiple groups, consumers which use the WS-FTA Proxy issue requests only to group leaders, which forward the request to the other replicas within their groups. This strategy reduces network traffic and turns the knowledge of the whole group membership unnecessary for the customer. For legacy consumers, the fact that the service is replicated and that one or more groups exist is fully transparent, since just one replica - a group leader - interacts with this consumer.

Figure 2 illustrates the procedure necessary for the reliable execution of a service request. The first step consists in sending the request to all group leaders (Step 1). Legacy consumers send the request to just one leader (Step 1a) and this leader forwards the request to the remaining leaders (Step 1b). The communication among replicas is done using the GC component, which provides reliable communication. The next step is the ordering of received requests, which is performed by group leaders. The MO component allows the use of different ordering algorithms. Since this process depends on the chosen ordering algorithm, its execution is not illustrated in the figure. After defined the execution order, all leaders send the requests in the established order to the other replicas in their groups (Step 2). For each group, if the replication technique adopted by this group requires all group replicas to process the request, the group leader receives through the GC component the results returned by each replica (Step 3). For consumers built using the WS-FTA Proxy, group leaders forward the results generated by replicas of the corresponding group, so that the proxy can execute a voting algorithm and select the result that will be delivered to the service consumer (Step 4). For legacy consumers, the leader that received the request is also responsible for receiving the results from all other group leaders (Step 4a), and through a voting algorithm elects the result that will be sent to the legacy consumer (Step 4b).

Given that a leader represents solely its group to the consumers, it is a point of failure for the whole group. Therefore, it is important that all other replicas in a group monitor the group leader, so that, in case of failure, some other replica takes over its role and notifies the other group leaders of this change. Ordinary replicas are also monitored, and in case of failure detection, recovery mechanisms are activated in order to try to restart the replica in the same host or, in case of host failure, to create a new replica on a different host. The current state is obtained from the LOG component, which receives from the group leader state updates periodically.

In the case of a group failure (e.g. when the connection with the external network fails), legacy consumers that have issued requests for this group will not have failure transparency. To keep using the service, legacy consumers will have to get a reference for other replica leader. To tolerate group failure, consumers can use the WS-FTA proxy.

The organization of the replicas in different groups makes possible the distribution of these groups in different local networks interconnected through a wide area networking infrastructure, e.g. the Internet. For example, it is possible to place one group in a local network in the U.S. and another group in a different network located in Europe, maintaining the service availability even in the presence of network faults. The interoperability among replicas in different networks is given by the adoption of a platform and transport-independent application level protocol - e.g., SOAP. Additionally, the message exchange must be reliable, given that these messages will guarantee the maintenance of a consistent state among service replicas. Message recovery may be either given by the adoption of WS-Reliability [12] or by implementing recovery mechanisms at application level.

Another advantage visualized in the separation of the replicas in different groups, is a major flexibility for covering different classes of failure [2], allowing a high independence degree among groups. In a distributed environment, this allows each group to be placed at a different network, and to use different replication techniques, transport protocols, and mechanisms for failure detection and recovery.

## 4. REFERENCE IMPLEMENTATION

In the previous section, WS-FTA is defined as an abstract description of components that supply dependability for Web Services. This level of abstraction allows the WS-FTA to be implemented on top of different platforms.

A reference implementation was developed in Java using the Java Development Kit (JDK), version 1.6. In this reference implementation, all components are represented as Java classes, interacting in accordance with the previously presented description.

The communication protocol is encapsulated by the GC component. The WS-FTA defines classes and interfaces that must be implemented to provide the communication among replicated services and architectural components. In this reference implementation, instead of adopting a group communication framework, the group communication mechanisms were implemented from scratch. In order to allow interoperability with other implementations of the architecture and with legacy systems, SOAP [1] was adopted as the base communication protocol for sending/receiving messages inside and outside groups. Moreover, the active replication technique was adopted for internal group organization.

One of the main goals of the communication among group leaders is to guarantee that consumer requests are executed with total ordering in all replicas. This guarantee is provided by the MO component using any total ordering algorithm. In this reference implementation, an optimistic total order algorithm, based on ideas presents in [10], was adopted. In this algorithm, a leader multicasts a request sending $n-1$ unicast messages, where $n$ is the number of the leaders. For each unicast message, a delay is calculated, trying to guarantee the delivery of messages totally ordered. This delay must be calculated according to the network communication time between sender and receiver. Assuming a set of leaders $L = \{A1, B1, C1\}$, where $A1$ send messages to $B1$ and $C1$, and the transmission time between $B1$ and $A1$ is $15ms$ and between $C1$ and $A1$ is $10ms$ - i.e., messages sent by $A1$ to $C1$ arrive before messages sent to $B1$. In order to compensate this difference, a delay is added before sending messages to $C1$. This delay is calculated by the following formula:

$$Ta_{(r \longrightarrow r')} = MaxTNode_{(r)} - T_{(r \longrightarrow r')}$$

Where:
$Ta_{(r \longrightarrow r')}$ : delay for sending the message from replica $r$ to $r'$;
$T_{(r \longrightarrow r')}$ : estimated time for the message sent by $r$ to be received by $r'$;
$MaxTNode_{(r)}$ : maximum $T_{(r \longrightarrow r')}$.

In the example, $MaxTNode_{(A1)} = 15ms$ and $Ta_{(A1 \longrightarrow C1)} = 5ms$, i.e., messages sent to $C1$ will be delayed by $5ms$.

Besides ordering requests, it is also necessary to detect the failure of replicas. This task is performed by the RFD and GFD components of the WS-FTA architecture. RFD and GFD are implemented by class "FailureDetector", which provides two monitoring methods: Push and Pull. In the Push method, replicas send "I am alive!" heartbeat messages periodically, which inform the detectors that the replica is working properly. If the detector does not receive the message within a specified time limit, the detector puts the replica under suspicion. This method requires the replica and the detector to agree on a frequency in which the heartbeat messages will be sent.

In the Pull method, the detectors periodically send "Are you alive?" messages to the replicas to check if they are working properly, and wait for the replica to send a response until a timeout expires. If the response is not received within this time limit, the detector suspects of replica failure.

In both methods, the detector waits for the response during a certain time. However, if the service does not answer, the detector assumes that the replica has failed and takes actions to maintain the reliability level. These actions include sending a notification to other detectors and to the REC component, which executes the faulty replica recovery process.

## 5. PERFORMANCE EVALUATION

This section presents some results obtained from tests executed with the reference implementation of the WS-FTA architecture, described in the previous section, in order to evaluate its performance.

Since WS-FTA allows the use of different replication techniques, ordering algorithms, failure detection algorithms and group communication protocols, it is necessary to identify which techniques, algorithms and protocols have been used in the tests. The reference implementation was configured with the following characteristics:

- Group communication protocol: SOAP;
- Replication technique: Active replication;
- Failure detection algorithm: Pull (in both RFD and GFD);
- Message ordering algorithm: Optimistic order (described in section 4).

These tests were executed on four computers with AMD $1.83GHz$ processors, $512MB$ of RAM, running Windows XP SP2. Three different configurations have been tested: one group with two replicas; two groups, each one with two replicas; and one group with four replicas. The replicas were deployed on different machines interconnected by a $100Mbps$

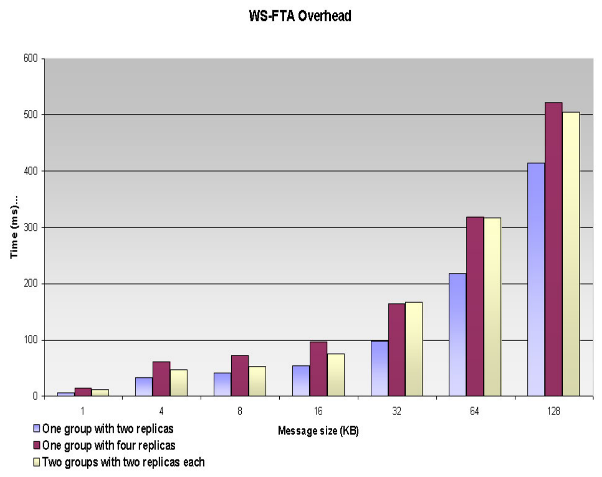

**Figure 3: Increase in response time**

Ethernet LAN. The same tests were executed with a non-replicated service, aiming to evaluate the impact on performance resulting from the use of the architecture.

Figure 3 shows the cost incurred to improve the reliability of the service using the architecture. These performance figure have been obtained with a service that just replies with a payload from 1 to $128KB$, which was invoked 1000 times in order to obtain the average response time. The obtained results show a $6ms$ overhead for accessing one group with two replicas, $14ms$ for one group with four replicas, and $11ms$ for two groups with two replicas each, compared to the response time of the same service without replication, when the service replies with $1KB$ messages. This shows that it is possible to obtain a sharp increase in the dependability without a heavy impact on the response time of the service. The increase in the message size results in a proportional increase in the overhead, due to the use of a voting algorithm to decide which response is sent to the consumer. The results also show that four replicas divided in two groups lead to an overhead approximately 16% smaller than if these replicas are organized in only one replication group, showing that the use of more than one replication group can improve the performance of the replicated services.

## 6. CONCLUSIONS AND FUTURE WORK

This paper presented a software architecture for dependable Web Services. This architecture, called WS-FTA, provides an abstract description of all the necessary components for guaranteeing reliability and dependability in Web Services. The architecture is fully compatible with the current standards and specifications adopted by the Web Services technology.

Moreover, WS-FTA allows the formation and development of groups of service replicas using different technologies. The possibility of allocating groups in different local networks provides a higher degree of reliability, allows more flexibility and results in the confinement of network traffic within the local network. In addition, the division of replicas into groups makes possible the provision of fault tolerance for legacy consumers.

Inter-group communication occurs only among group leaders, reducing traffic in the wide area network, in which the response time is a limiting factor for distributed applications.

There are several new features planned to be incorporated to the WS-FTA architecture, such as: to provide quality of service (QoS) mechanisms for balancing load among replicas and for guaranteeing a limited response time; to allow the dynamic discovery of new service groups; to incorporate support for Web Services running on computational grids; and to implement various message ordering and failure detection algorithms, allowing the creation of groups able to tolerate a wide range of classes of failures that may occur in distributed systems.

## 7. REFERENCES

[1] E. Cerami. *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI and WSDL*. O'Reilly, 2002.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 1996.

[3] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys*, 2001.

[4] C.-L. Fang, D. Liang, F. Lin, and C.-C. Lin. Fault tolerant web services. *Journal of System Architecture*, 2006.

[5] W. He. Recovery in web service applications. In *IEEE International Conference on e-Technology, e-Commerce and e-Service*, Taipei, Taiwan, 2004.

[6] P. Jalote. *Fault tolerance in distributed systems.* Prentice-Hall, Inc, Upper Saddle River, NJ, USA, 1994.

[7] W. Li, J. He, Q. Ma, I.-L. Yen, F. Bastani, and R. Paul. A framework to support survivable web services. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, Colorado, USA, 2005.

[8] J. Salas, F. Perez-Sorrosal, M. Patiño-Martinez, and R. Jímenez-Peris. Ws-replication: A framework for highly available web services. In *15th International World Wide Web Conference*, Edinburgh, Scotland, 2006.

[9] G. T. Santos, L. C. Lung, and C. Montez. Ftweb: A fault tolerant infrastructure for web services. In *9th IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, Enschede, Netherlands, 2005.

[10] A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, Suita, Japan, 2002.

[11] P. Veríssimo and L. Rodrigues. *Distributed Systems for System Architects.* Kluwer Academic Publishers, 2001.

[12] WS-Reliability. *WS-Reliability Specification Version 1.1.* OASIS, 2004.

[13] L. Xu and J. Bruck. Deterministic voting in distributed systems using error-correcting codes. In *IEEE Transactions on Parallel and Distributed Systems*, volume 09, 1998.

[14] X. Ye and Y. Shen. A middleware for replicated web services. In *IEEE International Conference on Web Services (ICWS'05)*, Orlando, Florida, USA, 2005.