An Architecture for Fault Tolerant and Service-based Business Processes

Diego Zuquim Guimarães Garcia[1], Maria Beatriz Felgar de Toledo[1]

[1]Institute of Computing – State University of Campinas
PO Box 6176 – 13.084-971 – Campinas – SP – Brazil

{diego.garcia,beatriz}@ic.unicamp.br

Abstract. *A Business Process Management System (BPMS) supports business processes and organizations depend on its availability. Web services have been pointed as a suitable technology for BPMSs. Thus, the inclusion of fault tolerance in the Web service architecture is essential for process continuity. The goal of this paper is to propose a fault tolerant Web service architecture to be used with BPMSs. The architecture provides service mediation and monitoring, and uses Web service standards. The main contribution of this paper is an architecture to support fault tolerance for business processes based on Web services.*

1. Introduction

Typically, Business Process Management Systems (BPMSs) are used for critical activities by many organizations [van der Aalst et al. 2003]. Recently, Web services are becoming important components for business processes. Thus, the availability of a business process becomes dependent on the availability of the Web services used in the business process.

Fault tolerance is an important requirement for critical systems. However, currently there is not a standard for fault tolerance in Web services, such as in other technologies, for instance, CORBA (Common Object Request Broker Architecture) [Object Management Group 2004].

Considering this deficiency, several proposals for fault tolerance in Web services are being developed. However, in the proposals, Web services are not considered as a BPMS technology. Moreover, they do not deal with important requirements, such as the use of mechanisms provided by Web service standards that support simplicity and interoperability.

This paper focuses on the area of fault tolerance for Web services. The main goal is to propose an architecture that offers business process continuity even in the presence of faults, by means of the high availability of Web service components.

The architecture extends the Web service architecture and includes two new components: broker and monitor. The broker component is responsible for interacting with Web service registries and managing replicas (Web services with similar functionality). The monitor component is responsible for monitoring services and detecting errors. Furthermore, the standard Web service registry is extended for including QoS information about Web services, in addition to functional information [Lee et al. 2005].

The proposed architecture enables BPMSs to use Web services in business processes to support the cooperation among organizations. Moreover, it uses the basic Web service architecture to guarantee compatibility with Web service standards [Fan and Kambhampati 2005].

Another important characteristic of the architecture is the fault tolerance. This characteristic is implemented by the inclusion of mediation and monitoring mechanisms to support the use of service replicas.

The rest of the paper is organized as follows. Section 2 describes basic concepts, including Business Process Management Systems, Web Services and Fault Tolerance. Section 3 discusses the proposed fault tolerant architecture for Web service-based BPMSs. Section 4 presents implementation aspects and a performance evaluation of the architecture. Section 5 discusses related work. Section 6 presents conclusions and future work.

2. Basic Concepts

This section presents some basic concepts for the better understanding of this paper.

2.1. Business Process Management Systems

Business Process Management Systems (BPMSs) support the control of business processes. A business process is a collection of activities. Activities are descriptions of work pieces that collectively realize a business goal, typically within the context of an organizational structure. Differently from traditional Workflow Management Systems (WfMSs), BPMSs focus on the management of dynamic interorganizational processes. These processes comprise activities that may be performed by different business partners [Hollingsworth 2004, Woodley and Gagnon 2005].

BPMSs support Business Process Management (BPM). The life cycle of BPM (Figure 1) starts with process modeling (Design phase). The process is then registered with a BPMS, which can create process instances (Configuration phase). The BPMS coordinates instance execution and records information collected during execution (Enactment phase). The execution history is analyzed and used to improve process models (Diagnosis phase) [van der Aalst et al. 2003].

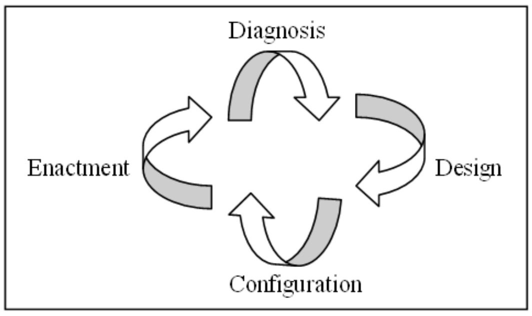

Figure 1. Business Process Management (BPM) life cycle

2.2. Web Services

A Web service-based BPMS supports the management of business processes that use Web services to achieve business goals [Leymann et al. 2002].

The Web service technology is based on the Service Oriented Computing [Papazoglou and Georgakopoulos 2003]. A Web service is an electronic service identified by a URI (Uniform Resource Identifier). XML (eXtensible Markup Language) standards are used to specify service interfaces and to invoke services through the Web. Figure 2 shows the Web service architecture [Booth et al. 2004]. The Web service technology comprises three basic standards [Alonso et al. 2004]:

- Web Services Description Language (WSDL): provides a model and a format for specifying the abstract functionality of a Web service as well as the concrete details of a service specification [Chinnici et al. 2005];
- Universal Description Discovery & Integration (UDDI): offers a Service Registry (Figure 2) that supports the publication and discovery of Service Providers, the Web services they make available and the technical interfaces that Service Consumers may use to bind and interact with the Web services [Clement et al. 2004];
- SOAP (formerly Simple Object Access Protocol): is a protocol intended for exchanging structured information in a decentralized, distributed environment, such as a Web service environment [Mitra 2003].

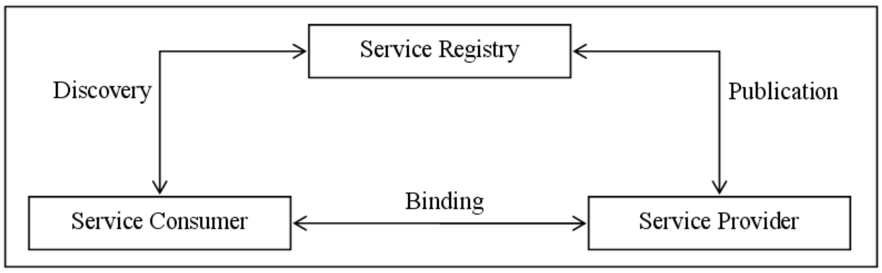

Figure 2. Basic Web service architecture

2.3. Fault Tolerance

System dependability is the ability of avoiding service failures that are more frequent and severe than the acceptable [Avizienis et al. 2004].

System dependability depends on some QoS aspects provided by the system. Dependability includes the following attributes: availability, reliability, safety, integrity and maintainability.

Fault tolerance is a means to achieve dependability. It may be defined as the act of avoiding service failures in the presence of faults. Frequently, the redundancy mechanism is used for implementing fault tolerant techniques. A system is able to tolerate a fault if it employs a type of redundancy [Gartner 1999].

Fault tolerant techniques may be classified according to the application phases: error detection, confinement, error recovery and fault treatment [Anderson and Lee 1981].

3. A BPMS Architecture

In this section, the proposed BPMS architecture is presented. It focuses on the necessity of evolving BPMSs for the inclusion of functionalities for the appropriate management of dynamic interorganizational processes. The architecture is based on the Web service architecture. It supports fault tolerance.

The proposed architecture includes components that are responsible for replica management, error detection and confinement. In the fault tolerant architecture, multiple replicas for the same Web service may be developed for different computational platforms and executed in different locations.

In the architecture, the configuration phase includes the publication of Web services. Service providers use UDDI structures to register replicas. The UDDI API (Application Program Interface) is used to provide the necessary data for the registration of Web service replicas into UDDI registries.

At the execution phase, a broker is used to create replica groups. After the creation of the replica group, the broker selects the Web service to be used. At this phase, a monitor is responsible for detecting errors. It verifies the status of the service during its execution. If an error occurs, the monitor interacts with the broker for the selection of a service replica.

Next, the architecture components are described and the modifications included into the basic Web service architecture are discussed.

3.1. Broker

The broker component is responsible for managing replicas. It creates Web service replica groups using the UDDI *tModel* concept. Web services are aggregated by means of specification sharing. Replica groups are dynamically defined and reflect the typical dynamic Web service environment, in which new Web services are continually offered and service offers are discontinued.

The broker uses the extended UDDI registry, which includes QoS information. Thus, the QoS demands of service consumers may be considered during the discovery of the replicas that will form a group.

Some properties of replica management may be defined in the broker component. For instance, it is possible to define the number of service replicas that form a replica group, in terms of the number of services of different providers and the number of access points of a service.

3.2. Monitor

In the proposed architecture, the monitor component is responsible for error detection, notification and confinement.

It monitors the execution of Web services by means of message interception [Baresi et al. 2006]. Moreover, it performs tests and analyzes service responses to detect errors.

The monitor notifies the broker if a Web service presents an error during its invocation or execution, for instance, if the service rejects an invocation, does not complete the execution or finalizes with an exception.

In the case of service errors after service selection, the monitor can transfer invocations to replicas discovered by the broker, with the purpose of guarantying the continuity of the business process.

The monitor component is responsible for obtaining from the consumer the required Web service type, along with the operation to be executed and the parameters required for the execution of the service. This component invokes the broker to perform the mapping of the service type into an available Web service replica.

Furthermore, the monitor can update the UDDI registry with the QoS information obtained during service monitoring.

3.3. UDDI Registry

The current UDDI standard is not very mature [Du et al. 2006]. For instance, it still lacks facilities for QoS description. In the proposed architecture, UDDI is extended to include QoS information, which is used by the broker during service selection and updated by the monitor. This extension enhances search flexibility. Searches can be based on functional characteristics and refined with required QoS attributes.

The UDDI information model [Clement et al. 2004] is composed of data structure types expressed in XML. The extended UDDI information model (Figure 3) aggregates the *qosPolicy* structure and its relationships. The data structure types are described below:

- *businessEntity*: top-level structure that contains descriptive information about a provider organization, such as contact and classification. Each *businessEntity* may provide various *businessServices*;
- *businessService*: represents a logical Web service that may have multiple implementations. It includes descriptive information about a Web service, such as name and classification;
- *bindingTemplate*: represents a service implementation and provides the information needed to bind with the service. Each *bindingTemplate* contains information such as access point and transport protocol;
- *tModel* (Technical Model): represents unique concepts in UDDI, such as namespaces and category systems. Examples include *tModels* based on WSDL and other documents that specify service interfaces;
- *publisherAssertion*: defines an association of *businessEntities*. It can be used by organizations to export their relationships, for instance subsidiary companies and industry consortia;
- *subscription*: describes a request to keep track of activities in a UDDI registry according to preferences provided with the request. Subscribers can register themselves for receiving information about changes in any of the UDDI structures;

- *qosPolicy*: represents a QoS policy for a Web service. The provided information describes attributes of a Web service represented by the *bindingTemplate* structure. References to *tModels* can be used to describe that a QoS policy conforms to particular specifications.

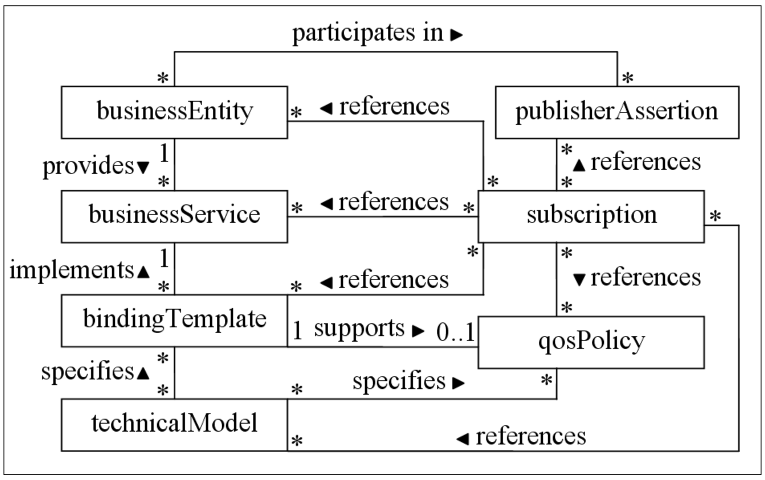

Figure 3. Extended UDDI information model

UDDI defines APIs [Clement et al. 2004] that standardize communication within and between UDDI implementations. APIs are grouped into sets. The extended UDDI registry includes extensions to the UDDI Inquiry, Publication and Subscription API sets to manipulate QoS information. Moreover, it uses the UDDI Security API set during the execution of operations in other sets.

In order to describe a service compatible with a set of specifications, references to the *tModels* that represent the specifications are included into the *bindingTemplate* structure. The *bindingTemplate* structures that refer to the same set of *tModels* are of the same type. Thus, *tModels* may be used as a base for replica group creation.

3.4. Component Interactions

The steps for service discovery and execution are presented in Figure 4. Subsequently, the steps are described.

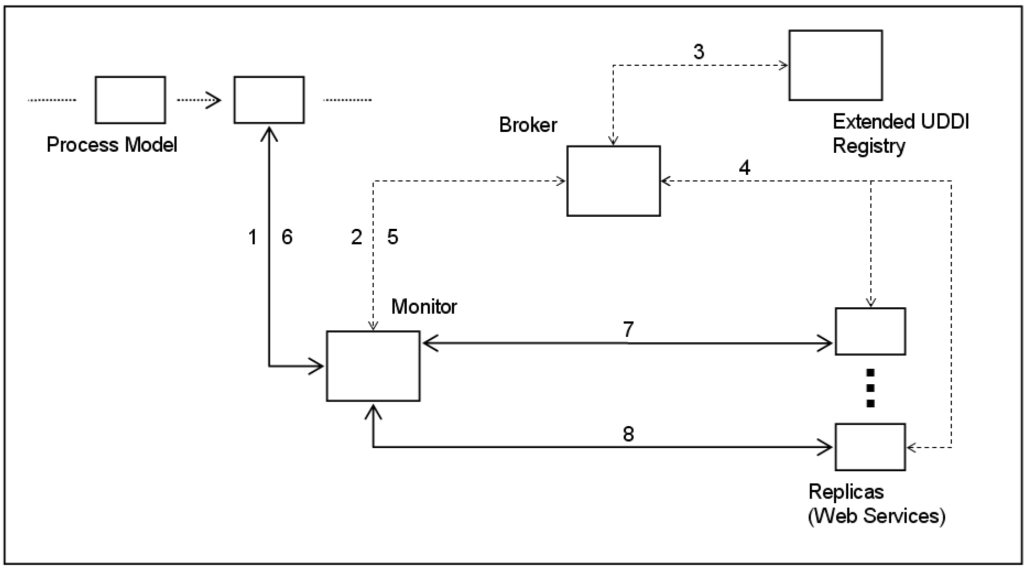

Figure 4. Fault tolerant Web service architecture for use with BPMSs

There is one monitor for each process activity. The broker is a remote component and, as well as the UDDI registry, a number of brokers can be used and deployed in different locations, such as along with monitors, UDDI registries or independently. These characteristics ensure autonomy in terms of the ability of defining and executing processes and using brokers.

The architecture enables BPMSs to use Web services as business process activities. Thus, it is possible to include services provided by different organizations into the business process of an organization.

At process design time, activities represent Web services. They indicate service types by means of *tModels*.

At process enactment time, when an activity realized by a Web service is reached, the BPMS requests a service that is able to implement the activity (Steps 1 and 2 in Figure 4).

A broker discovers Web services of the required service type and with the required QoS in the extended UDDI registry to create a replica group (Step 3). It performs availability tests for selecting the service to be used (Step 4). Subsequently, the broker returns the address of the selected service (Steps 5 and 6). Then, the BPMS may use the service through the returned access point.

At activity execution time, the monitor verifies the execution of the service (Step 7). If errors occur, the monitor requests the use of a service replica (Step 8). When service execution is finished, results are reported back.

4. Implementation and Evaluation

This section presents implementation aspects of the proposed architecture and discusses performance evaluation.

The architecture was partially implemented using: Sun Java Development Kit Version 1.5; Apache Axis Version 1.3 (providing SOAP and WSDL support); Apache

WS-Commons/Policy Version 0.9 (a WS-Policy implementation); and Apache Tomcat Version 4.1.24 (an application server).

The UDDI component was partially implemented using Apache jUDDI Version 0.9rc4, which is an open-source UDDI implementation compliant with the Version 2.0 specification that allows registries to be maintained for different purposes in different environments.

MySQL AB MySQL Version 5.0.16 was used to implement the UDDI database. The *QOS_POLICY* table stores Web service QoS attributes. This table refers to the *BINDING_TEMPLATE* table that stores Web service instances. It also refers to the *TMODEL* table containing technical models for QoS-related concepts.

In addition to directly using UDDI APIs through a SOAP API to interact with a registry, client APIs based on particular programming languages can be used. An extended client API for inquiry and publication operations based on Java was implemented. This API includes classes that represent UDDI elements. Developers can use it to access an extended registry without knowing about UDDI messages and structures. UDDI4J Version 2.0.4, a Java class library supported by HP, IBM and SAP, was used to implement the client API. UDDI4J has constructs that generate and parse messages sent to and received from a standard UDDI registry.

The extended UDDI is compatible with the basic UDDI and both types of UDDI registries can coexist in the same environment.

The architecture components were implemented as Web services. They interact through the Apache Axis SOAP engine. This approach allows the use of the architecture in restricted environments, such as intranets, and open environments, such as the Internet. Moreover, this approach offers the possibility of using brokers with different functionalities, according to consumer needs.

The replica management and error detection functionalities were partially implemented. The UDDI extension allows a refined replica group creation. Service replica selection is based on QoS, including the interoperability, response time and availability attributes. Service monitoring considers the response time attribute.

The goal of this partial implementation was to test the approach effectiveness. Some experiments were developed and the results are discussed below.

The evaluation focused on the performance impact that was generated by the inclusion of the fault tolerance support in the Web service architecture. In order to test the fault tolerance support, faults were introduced in a controlled manner during the experiments.

The tests were executed on a computational environment with the configuration shown in Table 1.

Table 1. System configuration

Operating system	Linux Fedora Core 2.6.16-1.2066_FC4
Java virtual machine	Sun Java Runtime Environment 1.5.0_06

Application server	Apache Tomcat 4.1.24
Database server	MySQL AB MySQL 5.0.16
SOAP engine	Apache Axis 1.3
UDDI registry	Apache jUDDI 0.9rc4 (extended)
Network	Ethernet / 100 Mbps
Processor	Intel Pentium 4A / 2800 MHz / 133 MHz
Memory	512 MB / 266 MHz
Motherboard / Chipset	Intel Sea Breeze D845GVSR / Intel Brookdale-G i845GV
Hard disk	Samsung SP0411N / 40 GB / 7200 RPM / Ultra-ATA 133

Some experiments were developed to measure the impact in terms of response time. Three situations were tested. Initially, the basic Web service architecture was evaluated and the fault tolerance support was not employed. In this case, the consumer discovers a Web service by means of the UDDI registry and invokes it directly. The second situation was similar, but the fault tolerance support was used. To evaluate the effectiveness of the fault tolerance support, in the third situation, faults were injected into the system.

The impact in terms of response time generated by the proposed fault tolerance support was measured by the comparison of the results of the second and third situations with the first situation.

Figure 5(a) shows the response time for the three cases. The average response time was approximately 68 ms, when the Web service architecture extensions were not used. When the extensions were used, the response time was approximately 108 ms in the case without faults and 138 ms in the case with faults.

A Web service executed with fault tolerance presented an average additional cost of 58.8 % in the response time, in opposition with the execution of the same service without fault tolerance. When faults were introduced, the additional cost was increased by 27.8 % in terms of response time of the same service in the case without faults.

In order to assess the overhead of the broker and monitor components, other experiments were executed. The overhead in terms of CPU usage was measured in two situations: with and without faults.

Figure 5(b) presents the CPU usage percentages. The average overhead introduced by the broker was approximately 13.8 %, and by the monitor, 13.5 %, in the situation without faults. With the introduction of faults, the average overhead was approximately 16.3 % for the broker, and 13.8 % for the monitor.

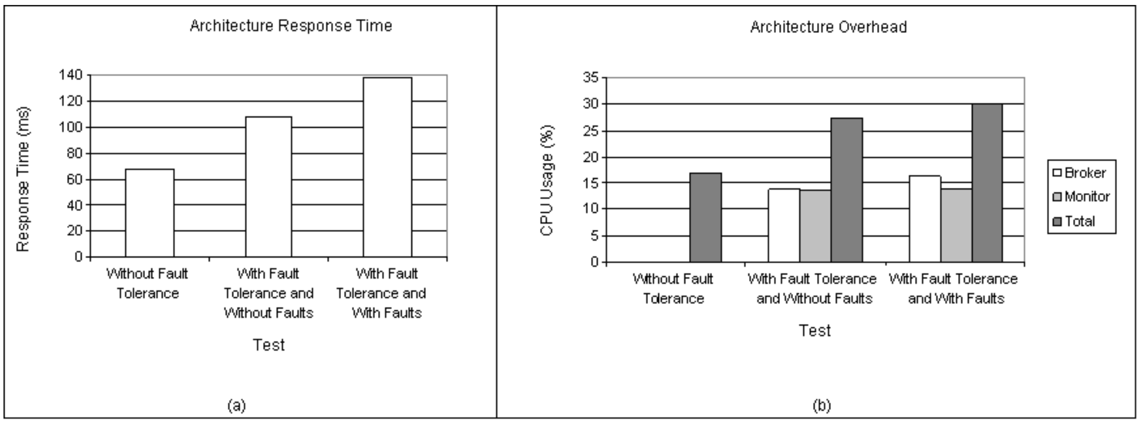

Figure 5. Results: (a) response time and (b) CPU usage of the fault tolerant architecture.

5. Related Work

This section presents some studies on fault tolerance in Web services. In the studies, the use of mediation in the Web service architecture is shared. However, they employ different fault tolerant mechanisms and techniques, such as passive replication, active replication, N-version model and checkpointing/rollback. Typically, these studies do not consider the use of Web service standard mechanisms and the application of Web services in the BPM area.

Dialani et al [Dialani et al. 2002] discuss that distributed computing and grid computing applications should be designed with fault tolerance to achieve robustness. However, the Web service community has not been focused on this aspect. This deficiency influenced some modifications in the SOAP layer with the goal of offering fault tolerance by checkpointing and rollback mechanisms. The proposed architecture for the development of fault tolerant Web services focuses on error recovery and, differently from the approach presented here, does not deal with the management of multiple service replicas.

Similarly to the approach proposed in this work, the passive replication technique is explored in [Liang et al. 2003]. To achieve fault tolerance, some modifications are proposed for the WSDL and SOAP standards with the purpose of allowing the specification of Web service replicas and the redirection of service requests, respectively. The approach presented here employs a mediation layer in the Web service architecture, which offers the necessary functionalities for managing replicas. Thus, it does not depend on modifications in the interface description language and the message interchange protocol of Web services. The approach includes a UDDI extension, but the extended UDDI is compatible with the standard.

In [Santos et al. 2005], the active replication technique is used in Web services to support consumer transparent fault tolerance. The architecture has a central forwarding component that includes the mechanisms responsible for managing replicas. It acts as a broker between Web service consumers and providers, as proposed in the architecture presented here. To deal with the fault point represented by the forwarding component, the architecture has a backup component. In the architecture proposed here, the broker is implemented as a Web service and enables consumers to use different brokers. Moreover, differently from the approach presented here, which uses UDDI *tModels* for creating service replica groups, the forwarding component has a configuration system for creating groups.

A N-version model implementation for Web services is presented in [Looker et al. 2005]. The N-version model is a design pattern for fault tolerance. In this model, in order to avoid errors caused by, for instance, specification and implementation problems, replicas are developed as different versions. In the proposal, the consumer stub is incremented for enabling consumers to provide service lists. Here, in a similar manner, replicas are Web services with similar functionality and the N-version model is supported by the loose coupling of the Web service technology. However, differently from the approach proposed here, the N-version model implementation demands the consumer to obtain the Web service status.

6. Conclusion

In this paper, a fault tolerant Web service architecture for use with BPMSs was described. Implementation aspects and a performance evaluation of the proposed architecture were presented. The approach extends the basic Web service architecture with the inclusion of broker and monitor components and a UDDI extension. The extensions introduce a significant additional cost, but the cost is acceptable because of the offered benefits. Business process continuity is offered by the fault tolerance support.

The main contributions of this paper are a fault tolerant architecture for Web service-based BPMSs and its partial implementation.

Future work includes the execution of additional tests to evaluate the approach in different scenarios. Moreover, the proposed extensions may be enhanced to handle a broader range of QoS attributes and to support mobile Web services. Providing services in mobile environments comprises QoS issues that are not included in fixed environments. Another direction is the inclusion of context-awareness facilities into the architecture to enable the delivery of services that adapt to different contextual dimensions. Furthermore, the absence of request redirection coordination generates an unstable system. A load balancing mechanism may be used to avoid the redirection of many consumer requests to the same replica.

References

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web services: concepts, architectures and applications*. Springer-Verlag.

Anderson, T. and Lee, P. A. (1981). *Fault tolerance - principles and practice*. Prentice-Hall, Englewood Cliffs.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Baresi, L., Guinea, S., and Plebani, P. (2006). WS-Policy for service monitoring. In Bussler, C. and Shan, M.-C., editors, *VLDB 6th International Workshop on Technologies for E-Services*, volume 3811 of *LNCS*, pages 72–83. Springer.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web services architecture, W3C working group note. Technical Report 11-Feb-2004, W3C. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, February.

Chinnici, R., Moreau, J.-J., Ryman, A., and Weerawarana, S. (2005). Web services description language, part 1: Core language, version 2.0, W3C working draft. Technical Report 10-May-2005, W3C. http://www.w3.org/TR/2005/WD-wsdl20-20050510/, May.

Clement, L., Hately, A., von Riegen, C., and Rogers, T. (2004). UDDI, version 3.0.2, UDDI spec technical committee draft. Technical Report 19-Oct-2004, OASIS. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm, October.

Dialani, V., Miles, S., Moreau, L., Roure, D. D., and Luck, M. (2002). Transparent fault tolerance for web services based architectures. In *Euro-Par'02: Proceedings of the 8th International Euro-Par Conference on Parallel Processing*, pages 889–898, London, UK. Springer-Verlag.

Du, Z., Huai, J., and Liu, Y. (2006). Ad-UDDI: An active and distributed service registry. In Bussler, C. and Shan, M.-C., editors, *VLDB 6th International Workshop on Technologies for E-Services*, volume 3811 of *LNCS*, pages 58–71. Springer.

Fan, J. and Kambhampati, S. (2005). A snapshot of public Web services. *SIGMOD Record*, 34(1):24–32.

Gartner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26.

Hollingsworth, D. (2004). The workflow reference model 10 years on. In *Workflow Handbook 2004*. WfMC, Winchester, UK.

Lee, E., Jung, W., Lee, W., Park, Y., Lee, B., Kim, H., and Wu, C. (2005). A framework to support QoS-aware usage of Web services. In *ICWE '05: Proceedings of the 5th International Conference on Web Engineering*, pages 318–327. Springer.

Leymann, F., Roller, D., and Schmidt, M.-T. (2002). Web services and business process management. *IBM Systems Journal, New Developments in Web Services and Electronic Commerce*, 41(2):198–211.

Liang, D., Fang, C.-L., Chen, C., and Lin, F. (2003). Fault tolerant web service. In *APSEC'03: Proceedings of the Tenth Asia-Pacific Software Engineering Conference*, pages 310–319, Washington, DC, USA. IEEE Computer Society.

Looker, N., Munro, M., and Xu, J. (2005). Increasing web service dependability through consensus voting. In *COMPSAC'05: Proceedings of the 29th Annual International Computer Software and Applications Conference Volume 2*, pages 66–69, Washington, DC, USA. IEEE Computer Society.

Mitra, N. (2003). SOAP, part 0: Primer, version 1.2, W3C recommendation. Technical Report 24-Jun-2003, W3C. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, June.

Object Management Group (2004). Common object request broker architecture: Core specification, version 3.0.3. Technical Report 12-Mar-2004, OMG. http://www.omg.org/cgi-bin/apps/doc?formal/04-03-12.pdf, March.

Papazoglou, M. P. and Georgakopoulos, D. (2003). Service-oriented computing. *Communications of the ACM*, 46(10):24–28.

Santos, G. T., Lung, L. C., and Montez, C. (2005). Ftweb: A fault tolerant infrastructure for web services. In *EDOC'05: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, pages 95–105,Washington, DC, USA. IEEE Computer Society.

van der Aalst, W. M. P., ter Hofstede, A. H. M., and Weske, M. (2003). Business process management: A survey. In *BPM'03: Proceedings of the 1st International Conference on Business Process Management*, pages 1–12. Springer.

Woodley, T. and Gagnon, S. (2005). Bpm and soa: Synergies and challenges. In *WISE'05: Proceedings of the 6th International Conference on Web Information Systems Engineering*, pages 679–688. Springer.