Policy Based Distributed Run Time Fault Diagnoser Model for Web Services

K. Jayashree[1] and Sheila Anand[2]

[1] Research Scholar
k.jayashri@gmail.com
[2] Dean(Research) Computer studies,
Rajalakshmi Engineering College, Chennai, India
sheila.anand@gmail.com

Abstract. Fault Management is one of the crucial areas in Web Service Management. Tools, methodologies and techniques that support and automate the management efforts have immense commercial application and patentability. In this paper, we present WISDOM (Web Service Diagnoser Model) a generic architecture for detecting faults in Web Services. The focus of WISDOM is on diagnosing problems that occur at run-time. It does not address issues relating to web service testing, debugging, specification or design faults. An independent and external Fault Diagnoser (FD) is used to detect run-time faults during publishing, discovery, composition, binding and execution of web services. We have tested the proposed model by using a sample web service application and present the details.

Keywords: Web Service Faults, Fault Diagnoser, Fault Diagnosis, Fault Diagnosis Model, Web Service Policy.

1 Introduction

Web Services have emerged as the most popular paradigm for distributed computing. The emerging paradigm of Web services opens a new way of quickly designing, developing and deploying web applications by integrating independently published Web services components. Web Service technology is being increasingly used in many practical application domains, such as groupware, electronic commerce, application integration , Transaction processing telecommunications, geographical information systems and digital imagery. Web services are based on XML standards and can be developed in any programming language. It facilitates the delivery of business applications as services accessible to anyone, anytime, at any location, and using any platform.

Managing web services and making them available have become critical issues. Fault Management is one of the crucial areas in Web Service Management. Traditional fault management tools cannot automatically monitor, analyze, and resolve faults in web Services. As Web services are distributed in nature, it is hard to trace the flow of transactions when a large number of services, supplier and

N. Meghanathan et al. (Eds.): CCSIT 2012, Part III, LNICST 86, pp. 9–16, 2012.

technologies collaborate together to perform an activity. This makes monitoring and Fault management of Web Service more difficult then centralized applications. The first step in fault management is to identify and classify the faults that may occur in services. Tools, methodologies and techniques need to be developed to support and automate the fault management efforts.

In this paper, we present WISDOM (Web Service Diagnoser Model) a generic architecture for detecting run-time execution faults in Web Services. The rest of the paper is organized as follows. Section 2 discusses the related work and Section 3 presents the proposed architecture for Web Services Fault Diagnosis. In Section 4, we have discussed the implementation of WISDOM using a sample web service application. Section 5 presents the conclusion and future work.

2 Related Work

In this section we discuss work related to web service monitoring, fault taxonomy and fault detection. There has been considerable work relating to web service testing, debugging and fault tolerance. We however present prior work related to web service monitoring and fault diagnosis.

Avizienis et al [1] discuss basic concepts about the threats to software dependability and security: failures, error and faults. They have classified the different types of software faults that may affect the system during its lifetime. They have presented eight basic viewpoints: phases of creation or occurrence, system boundaries, phenomena logical cause, dimension, objective, intent, capability and persistent. They have examined fault prevention, fault tolerance, fault removal, and fault forecasting. In our proposed work we have used the concepts relating to error detection and recovery explained with respect to fault tolerance to build and implement the Fault Diagnoser model.

Beth A.Schroder et al[2] describes correctness checking as the monitoring of an application to ensure consistency with a formal specification. Monitoring is used as a verification technique to detect runtime errors. The have described the monitoring system as an external observer that monitors a fully functioning application. It can provide increased robustness, security, fault – tolerance and adaptability. In our work we have applied the concept of monitoring and the external observer idea to design the Fault Diagnoser.

Delgado et al [3] have presented a taxonomy for runtime software fault-monitoring approaches for traditional software. Web service characteristics, however, differ considerably from that of traditional software. Web services are loosely coupled and dynamic in nature and monitoring of web services require additional issues to be addressed. Qi yu et al[7] describes web services are autonomous and loosely coupled and they interact dynamically without a priori knowledge of each other. Therefore failures in web service environment are expected to happen frequently. Thus a key step towards such a mechanism is to define what failures are in web service interaction environment and to provide a clear taxonomy for all these kinds of failures.

Stephan Bruing et al [5] have proposed fault taxonomy for possible faults in Service Oriented Architecture (SOA). Faults discussed include Hardware Fault, Software Fault, Network Fault and Operator Fault. They have demonstrated the application of the taxonomy with an example of travel agency. They have not implemented the taxonomy to detect faults in SOA. Farzaneth mahdian et al[15] present an approach to detect faults in the architecture level of service oriented systems which extends the formal SOA core meta model to support fault tolerance. In the fault tolerance architecture presented monitors are used to check the operation of different components of the system.

Benharref et al [4] have presented a web service based architecture for detecting faults in web services using an external Observer. Observers are web services published through Universal Description Discovery and Integration (UDDI). The observer is invoked by a manager which would provide specific information like the location of the service to be observed, its specification and when and what to observe. When the observer finds faults or the observation period expires it returns the result to the manager. This paper deals with only faults relating to inputs and outputs of the web services.

Mahbub and Spanoudakis[11] have developed a framework that monitor requirements for service based systems. The framework is applicable to service based systems whose composition process is specified in BPEL and specifies the requirements to be monitored using event calculus. Event logs are recorded during the execution of a service based system which is then checked against the specifications. Fabio Barbon et al [12] proposed a novel solution to the problem of monitoring web services described as BPEL processes. The approach allows for a clear separation of the service business logic from the monitoring functionality. The monitors they developed can check temporal, Boolean, time related and statics properties

Zheng Li et al [6] has developed a framework for monitoring the run time interactions of web services. Finite state automata are used to represent the service constraints. The framework involves interception of service messages and conformance checking with the service constraints. They have implemented the framework using Java-based open source platform for validating SOAP messages.

3 Proposed Architecture for Web Service Fault Diagnoser

In this paper, we present WISDOM (Web Service Diagnoser Model) a generic architecture for detecting faults in Web Services. The focus of WISDOM is on diagnosing problems that occur at run-time. It does not address issues relating to web service testing, debugging, specification or design faults. An independent and external Fault Diagnoser (FD) is used to detect run-time faults during publishing, discovery, composition, binding and execution of web services. The WISDOM architecture is given in Figure 1.

The Fault Diagnoser (FD) uses policies; a set of rules or guidelines to detect faults that occur during the execution of web services. A service provider will publish their services in the service registry and also provide guidelines for their web service execution. These guidelines are stored as policy in the policy database. For instance, one of the common run-time errors occurs in the specification of input parameters while invoking web services. The service provider would provide in the policy the

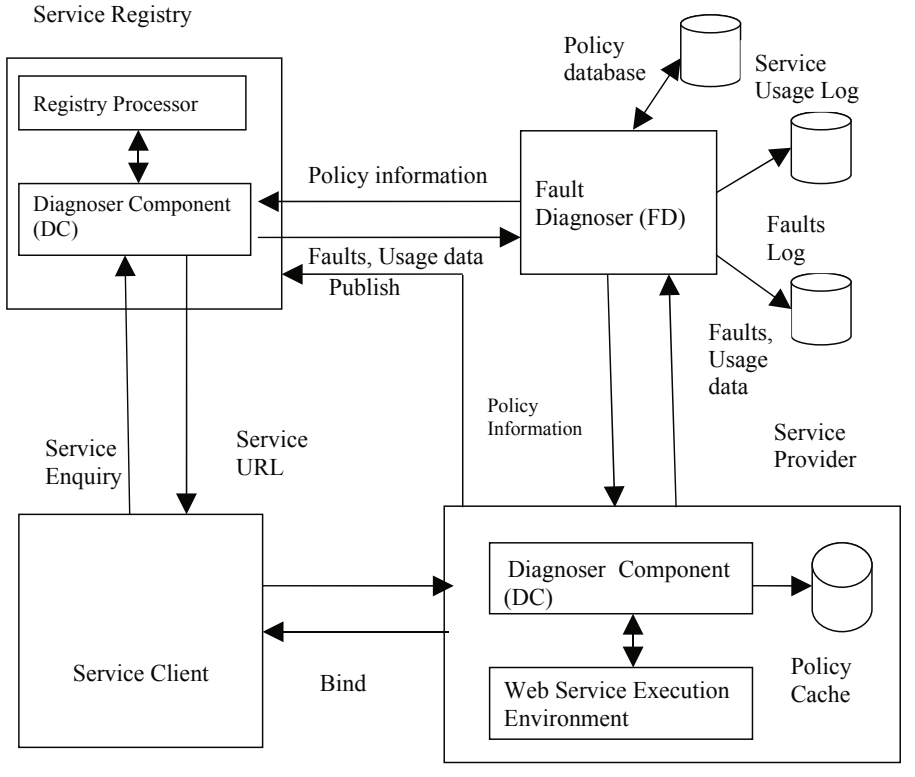

Fig. 1. WISDOM – Architecture for Web Service Fault Diagnosis

details of all run time parameters, their types and constraints. Some sample constraints that can be used to validate the numeric and alphanumeric input parameters could be:

Numeric Constraints:

- Constraint: Fraction Digits- Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
- Constraint: Maximum Inclusive - Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
- Constraint: Maximum Exclusive- Specifies the upper bounds for numeric values (the value must be less than this value)
- Constraint: Minimum Exclusive- Specifies the lower bounds for numeric values (the value must be greater than this value)
- Constraint: Minimum Inclusive - Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)

Alphanumeric Constraint

- Constraint: Data type – specifies whether the parameter is a string, list or matrix
- Constraint: Size- Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
- Constraint: Minimum Range - Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
- Constraint: Maximum Range - Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
- Constraint: Enumeration- Defines a list of acceptable values

Likewise the web service can provide policy information on output parameters that is results of execution of web services. A Diagnoser Component (DC) is present in every service provider to detect such run time faults. When a user invokes a web service, DC will obtain the corresponding policy information from FD. It will use the policy guidelines to monitor the web service execution and give meaningful error messages to the user. We recommend the use of a cache to store frequently accessed web services to improve the performance.

All faults encountered are reported by DC to FD and store in a fault database. The faults can be analyzed to determine common faults that occur during web service execution and try to provide proactive information to users to avoid such errors. The service usage database is used to store the usage of individual web services. An expert system can also be developed to assist in fault diagnosis.

Web services are published in the service registry by the service provider and users access the registry to discover web services. A DC is present in every registry to detect faults in publishing and discovery of web services. A registry listing of web service comprises of three elements, White Pages, Yellow Pages and Green Pages. White Pages is at the highest level and contains basic information about the providing company and its services. Yellow Pages organize services by industry, service type or geography. The Green Pages provide the interface and URL locations to find and execute a Web service. The policy for publishing web services would give details of all elements and constraints for checking the values of these elements. One such constraint could indicate, for instance, whether the element is mandatory or optional. Different policies may be specified for web services based on the type or usage of web services. When a service provider submits a request for publishing a web service, the DC would obtain the corresponding policy information from FD. It will use the policy guidelines to monitor the web service to diagnose faults in publishing.

To discover the service the service user queries the service registry for services by specifying the Business name and Service that meets its criteria. A DC present in the service registry can use policy guidelines to diagnose faults in the submitted query.

Policies are used to specify the accepted behavior of web services and deviations from accepted behavior can be diagnosed as faults. Policies provide a flexible and easy method of specifying the correct behavior of web services. Polices can be changed dynamically to cater to the changing needs of web services.

4 Implementation and Results

We have used IBM Rational Application Developer IDE for the implementation of WISDOM. jUDDI (Java Universal Description Discovery and Integration) is used to configure the web service registry. The web services are published in the jUDDI registry which is implemented using MySQL database.

We have taken a sample web service application from the Railway services domain as an example. Web services have been created for Train Enquiry, Ticket Availability, Reservation and Reservation Status Enquiry. Train Enquiry service allows the users to determine the list of trains to the intended destination. Ticket Enquiry allows users to find out whether seats or berths are available in the train of their choice. They can also obtain information about the number of seats available in the required class. A passenger books tickets using Reservation service by providing details which includes Date of Travel, Train Number, Train Name, Departing station, Destination station, Class, Name, Age etc. Successful ticket bookings will be assigned a unique PNR (Passenger Name Record) number. Reservation Status Enquiry requires the passenger to key-in the PNR number and the reservation status is displayed.

We have simulated different types of publishing, discovery, binding and execution faults which include:

- Publishing faults – incomplete description, described features missing etc.
- Discovery faults – required resource not existing, not listed in the lookup service
- Binding faults – authorization denied
- Execution faults – incorrect input parameter, incorrect results

Policies were used to define the correct behavior of these web services. The policies were implemented as a set of XML tags. The Fault Diagnoser model and Diagnoser components were developed to check the inputs and outputs to registry and web services. We were able to successfully diagnose the above mentioned faults. Policies provided a flexible method to detail the correct behavior of web services. It was easy to modify the policy details to reflect changes in the web service execution.

5 Conclusion and Future Work

Fault Management is one of the crucial areas in Web Service Management. We have presented in this paper a generic architecture for fault diagnosis in the run-time execution of web services. Policies have provided a flexible and simple method to represent the correct behavior of web services. We have tested the proposed model by using a sample web service application. As future work, we intend to extend the model to comprehensively diagnose all types of run-time or execution faults. We also intend to focus on standardizing the representation of the policies and develop a language for representing the policy assertions.

References

[1] Avizienis, A., Radell, B., Ladwehr, C.: Basic Concepts and Taxonomy of Dependable and secure Computing. IEEE Transactions on Dependable and Secure Computing 01(1), 33 (2004)

[2] Schroeder, B.A.: Online Monitoring: a Tutorial. IEEE Computer 28(6), 72–78 (1995)

[3] Delgado, N., Gates, A.Q., Roach, S.: A Taxonomy and Catalog of Runtime Software Fault Monitoring Tools. IEEE Transactions on Software Engineering, TSE-0209-1203

[4] Benharref, A., Clitho, R., Dssouli, R.: A Web Service Based Architecture for Detecting Faults in Web Services, 0-7803-9088-I/05/S20.00©2005 IEEE

[5] Bruning, S., Weibleder, S., Malek, M.: A Fault Taxonomy for Service-Oriented Architecture

[6] Li, Z., Jin, Y., Han, J.: A run time Monitoring and validation Framework for Web Service Interactions. In: Proceedings of the 2006 Australian Software Engineering Conference, ASWEC 2006 (2006)

[7] Yu, Q., Liu, X., Bouguettaya, A., Medjahed, B.: Deploying and managing Web services: issues, solutions, and directions. The VLDB Journal, 537–572 (2008)

[8] Robinson, W.N.: Monitoring Web Service Requirements. In: Proceedings of the 11th International Requirements Engineering Conference 1090-705X/03

[9] Madeira, H., Costa, D., Vieira, M.: On the emulation of software faults by software faults by software fault injection. In: Proceedings of the International Conference on Dependable Systems and Networks, pp. 417–426. IEEE (June 2000)

[10] Dingwall-smith, A., Finkelstein, A.: From requirements to monitors by way of aspects

[11] Mahbub, K., Spanousdakis, G.: Run Time Monitoring of Requirements for systems composed of web services: Initial implementation and evaluation experience. In: Proceedings of the IEEE International Conference on Web Services, ICWS 2005 (2005)

[12] Barbon, F., Traverso, P., Pistore, M., Trainotti, M.: Run Time Monitoring of Instances and classes Web Service Compositions IEEE ICWS06 Journal (2008)

[13] Alam, S.: Fault management of Web Services Thesis (August 2009)

[14] Papazoglou, M.P., van den Heuvel, W.-J.: Web Services Managemnet: A survey. IEEE Internet Computing (2005)

[15] Mahdian, F., Rafe, V., Rafeh, R., Zand Miralvand, M.R.: Considering faults in service Oriented Architecture: a graph Transformation based approach. IEEE (2009)

[16] Vierira, M., Laranjeiro, N.: Comparing web Services and Recovery in the Presence of Faults. In: ICWS 2007 (2007)

[17] Degwekar, S., Su, S.Y.W., Lam, H.: Constraint specification and processing in Web Services Publication and Discovery. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2004) (2004)

[18] Console, L., Fugini, M.: WS-DIAMOND: An approach to web services- DIAgnosability. In: MONitoring and DIAgnosis. EDs IOS Press, Amsterdam (2007)

[19] Chen, F., Rosu, G.: Towards Monitoring-Oriented Programming: a Paradigm Combining Specification and Implementation. Elseiver science B.V (2003)

[20] Ardissono, L., Console, L., Goy, A., Petrone, G.: Enhancing Web services with Diagnostic Capabilities. In: Proceedings of the Third European Conference on Web Services. IEEE (2005)

[21] Siblini, R., Mamour, N.: Testing Web services, 0-7803-8735-x-105-2005 IEEE
[22] Papazoglou, M.P.: Web services: Principles and technology. Prentice hall, ISBN 9780321155559
[23] Liu, L., Wu, Z., Ma, Z., Wei, W.: A Fault tolerant framework for Web Services. In: World Congress on Software Engineering. IEEE (2009)
[24] Liu, A., Li, Q., Huang, L., Xiao, M.: FACTS: A Framework for fault–tolerant composition of Transactional Web Services. IEEE Transactions on Services Computing
[25] Plebani, P., Pernici, B.: URBE: Web Service Retrieval Based on similarity Evaluation. IEEE Transactions on Knowledge and Data Engineering (2009)
[26] Arganda, D., Cappielo, C., Fugini, M.G., Mussi, E., Percini, B., Plebani, P.: Faults and recovery Actions for Self-Healing Web Services. In: WWW 2006, Edinburgh, UK, May 22-26 (2006)
[27] HP Openview Web services Management, Business White Paper, Hewlett Packard (November 2002)
[28] Nagappan, R., Skoczylas, R., Sriganesh, R.P.: Developing Java Web services
[29] http://www.apache.org
[30] http://w3schools.com