Proceedings of the 12th INDIACom; INDIACom-2018; IEEE Conference ID: 42835
2018 5th International Conference on "Computing for Sustainable Global Development", 14th – 16th March, 2018
Bharati Vidyapeeth's Institute of Computer Applications and Management (BVICAM), New Delhi (INDIA)

An Approach to Automate the Audit Testing of Web Services

Anwar Bari
Department of Computer Application, Integral University, Lucknow, India.
dranwarbari@gmail.com

Mohd. Faizan Farooqui
Department of Computer Application, Integral University, Lucknow, India.
faizan_farooqui2000@yahoo.com

A. A. Zilli
Department of Comp. Sc. & Engg, Lucknow University, Lucknow, India.
aazilli@gmail.com

***Abstract* – Web service is a piece of software easily available on the internet used to share functionality of organization. However, software undergoes maintenance and evolution activities. These can be due to the addition of new features, or corrective maintenance. Different types of changes may be the reason behind nonconformance of assumptions with integrator. Well-organized testing techniques are required which automatically verifies that the updated web services are in conformance with the requirements of the client or integrators. Audit testing is performed for making sure that new web service is also acceptable to the integrators. This paper provides an overview of audit testing of web services and the different types of changes occurring in web services. An approach to implement the framework for automating the audit testing of web service is also presented.**

Keywords – Audit Testing; Change Detection; Web Services.

I. INTRODUCTION

Web service is a piece of software easily available on the internet containing a collection of methods having predefined input and output types used to share functionality of organization. Service providers do not share internal source code or company policies to any third party for testing their web service. As web services are available without awareness of its internal coding details black-box testing is the only approach to test web services [2].

Upon discovering a service system integrator starts using it, taking interfaces and test cases as contracts between web service subscribers and providers. However, web services have to be modified regularly to satisfy changes in the requirements or to fix bugs. Changes in a web service can affect the system of its integrators. In this paper a discussion on the different types of changes occurring in web services has been carried out.

The coding of web services is independent of programming language and platform, only requirement is usage of Web Services Description Language (WSDL) as standardized XML interface description, Simple Object Access Protocol (SOAP) as standardized messaging protocol and Universal Description, Discovery and Integration (UDDI) [3].

The tester of web services is having information related to service interface in WSDL, so it is quite difficult to test web services. Hence in this paper a framework for testing and automating the audit testing of web services along with its implementation details has been discussed.

II. OVERVIEW OF AUDIT TESTING OF WEB SERVICES

Web service is an autonomous collection of methods bundled into a single package having predefined input and output accessible over the Internet. In order to fix errors or to enhance their functionality web services are usually updated. When services are updated, the user software or websites which incorporates these new services are required to be retested, so as to be sure that the already working functionalities are not affected because of updates and do not lead to adverse effects. Such testing is known as regression testing. Audit testing is a special type of regression testing having restriction on time and cost [4].

Due to security reasons or company policies the code of web service is not available to the user or client and hence

 ISSN 0973-7529; ISBN 978-93-80544-28-1

testing web service become very difficult. The interface of web service is described using WSDL, so to perform audit testing the following procedure is generally applied.

In the start source code is generated using WSDL which helps in the process of generating test cases and also in its execution.

For each individual method wrapper classes are also generated related to the operations present in the web service respectively. Using any test generation tool a test suite is generated for these wrapper classes.

This generated test suite is used with the generated source code [1]. The execution of this test suite will automatically call the web service by sending request messages [1].

The outputs received from the web service are then collected. Which is analyzed to show problems and manual inspection may be done on the same.

III. TYPES OF CHANGES OCCURING IN WEB SERVICES

Upon discovering a service system integrator starts using it, selection of services for integration depends on functional and non-functional requirements. The software system used for developing service goes through maintenance for bugs fixing (cope with problems arisen during the service usage) or evolution of the existing ones such as addition of new features to satisfy changes in the requirements.

The services may be re-tested from time to time to be sure that the service provides the required functionality and fulfills the Quality of Service requirements. Any change in a web service might affect the systems using the service. Here, the clients or testers should be aware of types of changes occurring in a web service so as to analyze the effect on their systems.

Two kinds of changes occurring in web service can be identified depending on the effects and side effects they might cause:

Shallow changes: In shallow changes the effects are confined to a service or clients of that service only.

Deep changes: In deep changes the effects are of cascading types which may extend beyond the clients of a service.

For proper interaction services must mutually arrive to a service level agreement deciding the specifications between the service provider and the service client.

Whenever a new version of web service is released the following different scenarios can happen:

- Change in the service functional behavior: the new version responds to inputs in a different way or there is a difference in exception handling of the service and thus behaves differently which may cause failures in the system.
- Change in the service non-functional behavior: if there is any change in service implementation, supporting hardware or network configuration it might modify the service non-functional behavior.

During modification if interface and/or specification is not changed it remains unknown to service client. The system integrator now unintentionally uses the updated service and the response of the service for some inputs, might be different from previous responses. As the current version of a service meets integrator requirements, future versions may not.

Services generally evolve while applying changes such as Structural changes, Business protocol changes, Policy induced changes and Operational behavior changes.

IV. WEB SERVICE AUDIT TESTING PROCESS

In our previous work a critical analysis of issues occurring in Audit Testing of Web Services [13] was performed where the drawbacks of regression testing was discussed and the need and challenges for audit testing of web services was bought into light [14]. Audit testing is useful for being sure that if any new service is introduced or old service is updated, then that service is correctly incorporated in the current system or application[11]. In this present work an approach to automate the implementation of proposed framework [16] has been bought forward for testing and auditing of web service.

The technique for audit testing is divided into modules. The testing technique starts by Code Generation module considering the input and output parameters, protocol bindings, message formats and supported operations [2] needed for communicating with the web services enumerated in the UDDI[2] using the WSDL [1] of a service, automatically generate code in Java by creating necessary stub for working as a client (service consumer) and send service requests to the skeleton (service provider).

The Axis utility, WSDL2Java, analyzes WSDL to generate the source code from a service provider's WSDL [1]. Axis [5] offers required Java code implementation of the SOAP protocol. In code generation module Axis is used to produce four classes or interfaces using the WSDL of web service. The classes are generated in Java respective to all methods in that class taking into account the input and output parameters contained within service. Then an interface in Java for each port type defining the connection point is generated. For all bindings, a stub class describing the message format and protocol is generated. The binding element has attribute name describing name of the service and a type attribute describing the port of the binding. Now codes for service interface and generated. The service input output parameters defines the visible web services. These input output parameters assists in calling web service for testing purpose [1]. Finally generate wrapper classes which cover every method to a respective existing service operation. Wrapper class allow calling of

available service. This wrapper class will be passed on to any one of the test generation tool, which generates unit test.

The next module in the framework is Test Generation module which is inputted with the classes generated by code generation module. Test generation module examines the kind of information available in Java classes and creates a code portion which builds different types of calls to the web service for testing the performance of all procedures contained in the web service. All such unit test cases are collected to form a validation suite that tests the services pointed by the WSDL [1]. Required test cases are acquired by analyzing and converting test suites which are created for the system using the service's features [12]. It is required that any expression part of an assertion needs to be calculated and decoded into a literal.

The code portions along with the unit tests generated are called for providing inputs to all supported operations in a web service under test [2]. The supporting operation to be tested is within the wrapper class. For each service in WSDL a method is declared and for each input parameter in that service a method argument is designated [1]. Test generator produces unit tests which pass the required input parameters in order to call the web service.

This methodology can function independent of the tool used for test generation [7].

JCrasher[8] observes the kind of information from a collection of Java classes and creates portion of programs which creates calls of different types for validation of behavior of methods.

JCrasher tests errors by attempting the web service to be tested to crash. JCrasher transitively analyzes methods, finds out the scope of each tested method's parameter-space and selects parameter combinations and therefore test cases at random, considering the time given for testing it determines heuristics for deciding if any Java exception should be taken as a coding error or the assumptions taken by programmer have been violated by JCrasher supplied inputs.

After creation of test cases the Test Execution module executes the generated validation suite on the wrapper class with the stub code [1]. Each generated test case calls the supporting operation in the wrapper class. These supporting operations have been coded in such a way to pull the generated stubs which send SOAP requests to the skeleton to be tested [1].

JUnit [10] automates the execution of test cases and facilitates the collection of test case results, but does not offer any assistance in implementing test cases. JUnit [2] invokes the unit tests corresponding to the generated wrapper class. JUnit is used to run a unit-test from the validation suite as regression testing framework against the wrapper class for testing [9]. If any failure occurs this execution of validation class can throw an exception [12].

The process of generating service test cases from JUnit test suites can be completely automatic, or user-guided. In the first case, the JUnit test suite is translated so that operation invocations are left symbolic, whilst other expressions are evaluated and translated into literals. In the second case, the user can select the JUnit test cases that should be considered to generate service test cases.

The Response Analysis module consists of Monitor and Analyzer, whenever the service is invoked the consumer diverts the request to response analysis module, where the monitor intercepts the messages between service requester and web service in recessive method, transform the intercepted message into standard format, request is noted and redirected to the provider. Also collect the responses or error condition returned to message logs and redirects the same to the consumer. The monitor module works as a man-in-the-middle in order to collect web service responses, between the consumer and the provider [1].

The analyzer module is used to analyze the large number of request-response pairs to judges whether the component of web service meets the requirement of the standards and upon occurrence of an exception the analyzer module indicate problem and shortlists those test cases which might show errors in a conformance report.

This conformance report may be presented for manual inspection to determine if it is triggered by an error in the code of web service or the inputs given did not follow the service provider's assumptions [1]. Such execution of the web service must return useful error message for the identification and notification to clients or integrators regarding change in web service.

Lastly measures for selection [6], minimization and prioritization[14] of audit test cases may be applied to web services based on the conformance report. In our previous work a critical analysis of issues occurring in Audit Testing of Web Services [13] we discussed the schemes for selection, minimization and prioritization.

V. CONCLUSIONS AND FUTURE WORK

We have given an overview of audit testing of web services where code is generated from WSDL. Unit test cases are generated using automated unit test generation. For testing unit test cases are executed on the code.

While being used, a service can change its behavior or its non-functional properties, and the integrator may be not aware of such a change. Thus a discussion on the type of changes occurring in web services has been done. Changes in a web service interface typically affect the systems of its subscribers. Therefore, it is essential for subscribers to recognize which types of changes occur in a web service interface in order to analyze the impact on his systems.

Modifications or evolution in web services is happening

quickly due to security reasons or changes in company policies. Thus services may be re-tested from time to time to be sure that the service provides the required functionality as desired by the client. Audit testing is done for being convinced that updated services fulfill the Quality of Service requirements.

Concerning this an approach for implementing a framework has been discussed for testing and auditing of web services. In future work, the implementation of this approach may be tested and improvised and its efficiency in comparison to available techniques needs to be checked.

REFERENCES

[1] Evan Martin Suranjana Basu Tao Xie, "Automated Testing and Response Analysis of Web Services", IEEE International Conference on Web Services (ICWS 2007), 07/2007.

[2] E. Martin, S. Basu, and T. Xie, "Automated robustness testing of web services", In Proceedings of the 4th International Workshop on SOA and Web Services Best Practices (SOAWS *2006)*, October 2006.

[3] Masood, Tehreem, Aamer Nadeem, and Shaukat Ali. "An automated approach to regression testing of web services based on WSDL operation changes", 2013 IEEE 9th International Conference on Emerging Technologies (ICET), 2013.

[4] Hong Zhu, Member, IEEE Computer Society, and Yufeng Zhang, "Collaborative Testing of Web Services" IEEE transactions on Service Computing, Vol. 5, No. 1, January-March 2012

[5] Apache. Axis. http://ws.apache.org/axis/

[6] Ruchika Malhotra, Arvinder Kaur and Yogesh Singh, "A Regression Test Selection and Prioritization Technique", Journal of Information Processing Systems, Vol.6, No.2, pp.235-252, Jun 2010.

[7] K. Ashok, G. Kumar, A. Dhawan, and A. Dharani, "Automated regression suite for testing web services," in Int. Conf. Software Maintenance (ICSM), November 2009, pp. 590–592.

[8] C. Csallner and Y. Smaragdakis. "JCrasher: an automatic robustness tester for Java. Software: Practice and Experience", 34:1025–1050, 2004.

[9] Agitar. Agitar Agitatior 2.0, Novermber 2004. http://www.agitar.com/

[10] E. Gamma and K. Beck. JUnit, 2003. http://www.junit.org

[11] M. D. Penta, M. Bruno, G. Esposito, V. Mazza, and G. Canfora, Test and Analysis of Web Services - Web Services Regression Testing. Springer Berlin Heidelberg, 2007.

[12] Tao Xie. "WebSob: A Tool for Robustness Testing of Web Services", 29th International Conference on Software Engineering (ICSE 07 Companion), 05/2007.

[13] Anwar Bari, A. A. Zilli and S. Qamar Abbas. "Critical Analysis of Issues in Audit Testing of Web Services" International Journal for Scientific Research and Development 4.8 (2016): 645-650.

[14] C. D. Nguyen, Alessandro Marchetto, Paolo Tonella, "Challenges in Audit Testing of Web Services" 2011 Fourth International Conference on Software Testing, Verification and Validation Workshops

[15] Cu D. Nguyen, Alessandro Marchetto and Paolo Tonella, "Test Case Prioritization for Audit Testing of Evolving Web Services using Information Retrieval Techniques," In.proc.of the IEEE International Conference on Web Services (ICWS), Washington, DC, pp.636 - 643, 2011.

[16] Anwar Bari, Mohd. Faizan Farooqui, A.A. Zilli, "Study of Automated testing of Web services" Accepted in 5th International Conference on "Computing for Sustainable Global Development", 14th – 16th March, 2018, New Delhi (INDIA)

