International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol. 3, Issue. 4, Jul - Aug. 2013 pp-2102-2107 ISSN: 2249-6645

Testing of web services Based on Ontology Management Service

J. Mahesh Babu[1], P. Rajarajeswari[2], Dr. A. Ramamohan Reddy[3]
[1](M. Tech, Computer Science and Engineering, Madanapalle Institute of Technology & Sciences, Madanapalle, A.P,India)
[2](M. Tech[Ph.d], Asst. Prof., Deptt. of CSE, Madanapalle Institute of Technology & Sciences, Madanapalle, A.P, India)
[3](M. Tech, Ph.d, Professor and Head, S. V. University, Tirupathi, A.P, India)

***Abstract*:** *Web services are emerging technologies that can be considered as the result of the continuous improvement of Internet services due to the tremendous increase in demand that is being placed on them. They are quickly developing and are expected to change the paradigms of both software development and use, by promote software reusability over the Internet, by facilitate the covering of underlying computing models with XML, and by providing various and complicated functionality fast and flexibly in the form of composite service offerings. In this paper, create one web application and Framework for Web services (WS). Web application side any link fault means Web services to display fault link in background side. But proposed work each and every communication link to be checking process using JUnit tool. JUnit tool is net working tool that process is testing for Framework Web Services (WS).*

***Key Words*:** *JUnit Testing Tool, JUnit Test case, distributed/internet based software engineering tools and techniques, testing tools, web services.*

I. INTRODUCTION

Web Services (W3C, 2004b) are considered a new paradigm in building software applications; this paradigm is based on open standards and the Internet. Web Services facilitate the interconnection between heterogeneous applications since it is based on XML open standards that may be used to call remote services or exchange data.Web Services are considered an implementation or realization of the Service-Oriented Architecture (SOA) (Singh & Huhns, 2005), which consists of three roles: *Service Requester* (Consumer), *Service Provider*, and *Service Publisher*(*Broker*). To implement SOA, Web Services depend on a group of XML-based standards such as Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Universal Description, Discovery and Integration(UDDI). A problem that limits the growth of Web Services is the lack of trust worthiness by the requesters of Web Services because they can only see the WSDL document of a Web Service, but not how this service was implement by the provider. An example of using a Web Service is when building an application that needs to get information about a book (e.g., price and author) given the book's ISBN. Amazon provide a Web Service (see Cornelius, 2003) to fulfill this requirement and using the approach in this chapter it can assess how robust the service is before using it. Software Testing is mainly used to assess the quality attributes and detect faults in a software system and demonstrate that the actual program behaviour will conform to the expected behaviour. Testing techniques can be divided into black box and white box depending on the availability of the source code; if test data are generated depending on the source code, then a testing technique belong to white box, while if the source code is unavailable, then a testing technique belongs to black box.

This chapter's approach of Web Services testing assumes that the tester only have the WSDL document of the Web Service under test and not the source code, for this reason black box testing techniques will be used. It will also help the requesters to choose between Web Services doing the same task. However, Web Services testing still face many problems like unavailability of the source code to the requesters and that the traditional testing techniques do not cope with the new characteristics introduced by Web Services standards (Zhang & Zhang, 2005). This chapter introduces an approach to solve part of these problems, which is based on analyzing WSDL documents in order to generate test cases to test the robustness quality attribute of Web Services.

1.1 Generation of testbed. A service often relies on other services to perform its function. However, in service unit testing and also in progressive service integration testing, the service under test wants to be divorced from other services that it depends on. Techniques have been developed to produce service stubs or mock services to replace the other services for testing.

1.2 Checking the correctness of test outputs.

Research work has been reported in the literature to check the correctness of service output against formal specifications, such as using metamorphic relations, or voting mechanism to compare the output from multiple equivalent services etc.

II. Web services architecture

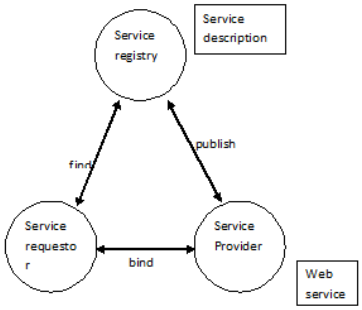


Fig-2: Web services architecture

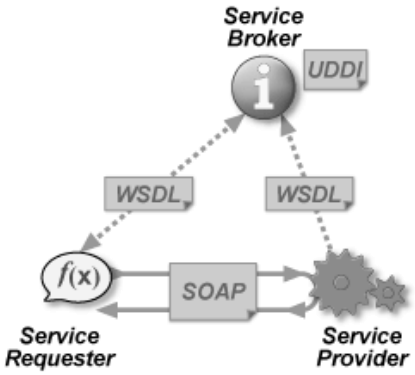


Fig-2.1: Architecture

Fig-2.1Although it is important, the centralized service registry is not the only model for Web service discovery. The simplest form of service discovery is to request a copy of the service description from the service contributor. After delivery the request, the service contributor can simply e-mail the service description as an attachment or provide it to the service requestor on a transferable media, such as a diskette. Between these two extremes, there is a need for a distributed service discovery method that provides references to service descriptions at the service provider's point-of-offering.

2.1WS-Inspection overview

The WS-Inspection specification does not define a service description language. Within a WS-Inspection article a single service can have more than individual reference to a service description. References to these two service descriptions should be put into a WS-Inspection document. If many references are available, it is useful to put all of them in the WS-Inspection document so that the document consumer can select the type of service description that they are capable of understanding and want to use. Fig-2.2 provides an overview of how WS-Inspection documents are used.

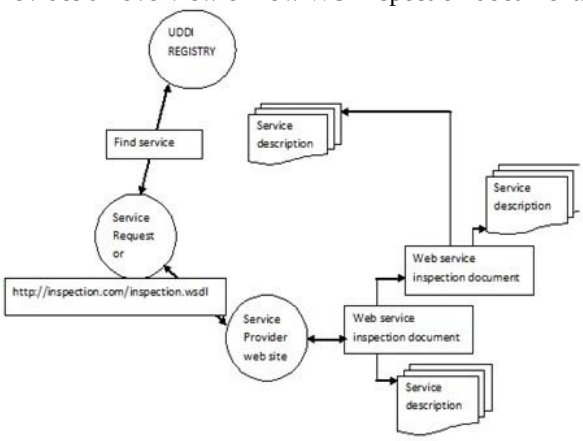


Figure 2.2: WS-Inspection overview

The WS-Inspection specification contains two primary functions, which are discussed in more detail in the next two sections.

2.2 The Architectural Models: The Service Oriented Model focuses on aspects of service, action and so on. Fig2.3 While clearly, in any distributed system, services cannot be sufficiently realized without some means of messaging, the converse is not the case: messages do not need to relate to services.

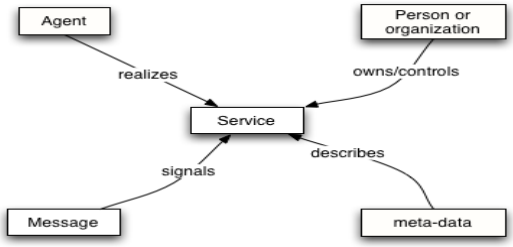
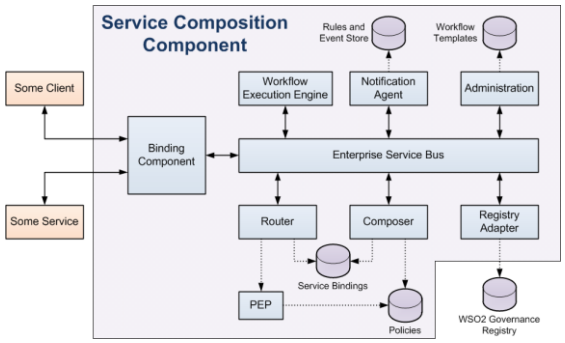


Figure 2.3. Simplified Service Oriented Model



The software infrastructure consists of an **Enterprise Service Bus** (Apache Service Mix), a BPEL Workflow Execution Engine (a customised version of Apache ODE), a Resource Registry (based on the WSO2 Governance Registry and accessed via a registry adapter) and a set of custom built centre components. These are:

- The Workflow Composer selects the concrete services provided by the Resource/Service Registry at runtime. In particular, it equipment a ranking mechanism to support this local selection step.
- It is the Router's responsibility to forward messages to the appropriate endpoints. For outbound messages, which address an endpoint just by semantic information, the router handles the service compulsory in addition to ensures the accessibility of suitable, dynamically created endpoints. For inbound messages it can be configured to route them via a policy enforcement point. It can also apply fault handling strategy in the case of message faults.
- The Notification Agent collects notifications and events from various components and produces higher level notifications according to a set of rules.

III. JUnit Testing Tool

It is important in the test driven development, and is one of a family of unit testing frameworks collectively known as xUnit. JUnit is a unit testing framework for the Java Programming Language.
JUnit promotes the idea of "first testing then coding", which importance on setting up the test data for a piece of code which can be tested first and then can be implemented. This approach is like "test a little, code a little" which increases programmer productivity and stability of program code that reduces programmer stress and the time spent on debugging.

JUnit Test Case:

A Unit Test Case is a part of code which ensures that the another part of code (method) works as expected. To achieve those required results quickly, test framework is required .JUnit is perfect unit test framework for java programming language.

A formal written unit test case is characterized by a known input and by an normal output, which is worked out before the test is executed. The known input should test a condition and the expected output should test a post condition.

There must be at least two unit test cases for each requirement: one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

Find defects. This is the classic objective of testing. A test is run in order to activate failures that expose defects. Generally, we look for defect in all exciting parts of the product.

Maximize bug count. The distinction between this and "find defects" is that total number of bugs is more important than coverage. We might focus narrowly, on only a few high risk features, if this is the way to find the most bugs in the time available.

Block premature product releases. This tester stops untimely delivery by finding bugs so serious that no one would ship the product until they are fixed. For every release result meeting, the tester's goal is to have new show stopper bugs.

Help managers make ship / no-ship decisions. Managers are typically concerned with risk in the field. They want to know about coverage (maybe not the simplistic code coverage data, but some indicator of how much of the product has been

addressed and how much is left), and how important the known problems are. Problems that show important on paper but will not lead to customer dissatisfaction are probably not relevant to the ship decision.

Minimize technical support costs. Working in combination with a technical support or help desk group, the test team identify the issue that lead to calls for support. These are frequently peripherally related to the product under test--for example, getting the product to work with a specific printer or to import data successfully from a third party database might prevent more calls than a low-frequency, data-damaging crash.
Assess conformance to specification. Any declare made in the specification is checked.

Conform to regulations. If a regulation specifies a certain type of coverage (such as, at least one test for every claim made about the product), the test group creates the suitable tests. If the regulation specifies a style for the specifications or other Documentation, the test group probably checks the style. In general, the test group is focusing on anything covered by regulation and (in the context of *this objective*) nothing that is not covered by regulation.

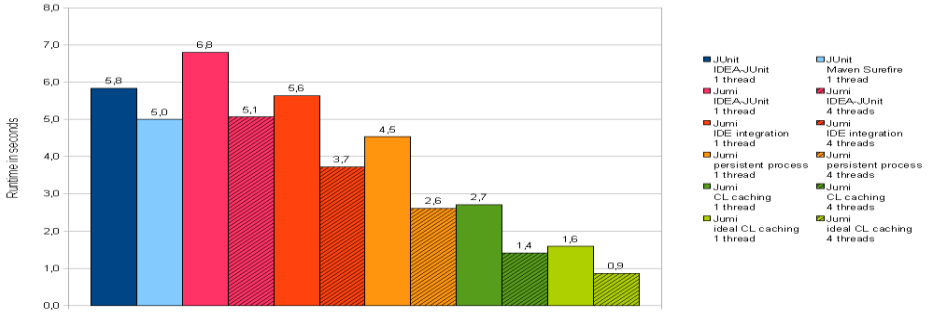
Minimize safety-related claim risk. Any error that could lead to mistake or injury is of primary interest. Errors that lead to loss of time or data or damage data, but that don't carry a risk of harm or damage to physical things are out of scope.

Find safe scenarios for use of the product (find ways to get it to work, in spite of the bugs). Sometimes, all that you're looking for is one way to do a task that will consistently work--one set of instructions that someone else can follow that will reliably deliver the benefit they are supposed to direct to. In this case, the tester is not looking for bugs.

Assess quality. This is a tricky objective because quality is multi-dimensional. The nature of excellence depends on the nature of the product. For example, a computer game that is rock solid but not entertaining is a lousy game. For example, *reliability* is not just about the number of bugs in the product. It is (or is often defined as being) about the number of reliability-related failures that can be expected in a period of time or a period of use. (*Reliability-related?* In measuring reliability, an association might not care, for example, about misspellings in error messages.) To make this prediction, you need a mathematically and empirically sound model that links test results to dependability. Testing involves meeting the data needed by the model. This strength involve extensive work in areas of the product believed to be stable as well as some work in weaker areas. Visualize a reliability model based on counting bugs found (perhaps weighted by some type of severity) per N lines of code or per K hours of testing.

Troubleshooting to make the bug report easier to understand and more likely to fix is (*in the context of assessment)* out of scope.

Verify correctness of the product. It is not possible to do this by testing. You can prove that the product is *not* correct or you can demonstrate that you didn't find any errors in a given period of time using a given testing strategy. However, you can't test exhaustively, and the product capacities fail under situation that you did not test. The best you can do is assessment--test-based estimation of the probability of errors.



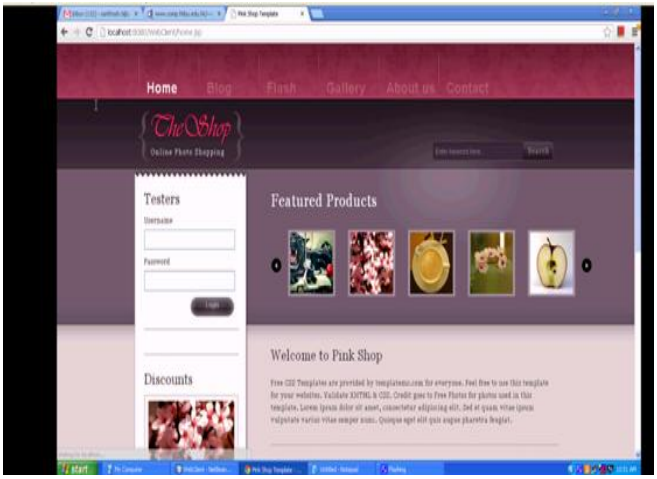
The time spent on 3.b only depends on the performance of the tester (s). It is irrelevant to the efficiency of the broker. Therefore, it is misplaced in our experiment. shows the average lengths of execution times on different tasks with the number of different types of subtasks ranging from 1 to 5. A quadratic polynomial figure fits the curve very well with R2 ¼ 0:9984.In summary, the experiments show that the broker is capable of dealing with test problems of practical sizes with respect to the number of testers registered, the size of the knowledge base, and the difficulty of test tasks.

IV. Testing Modulus:

4.1Test/Functional Service Generation:

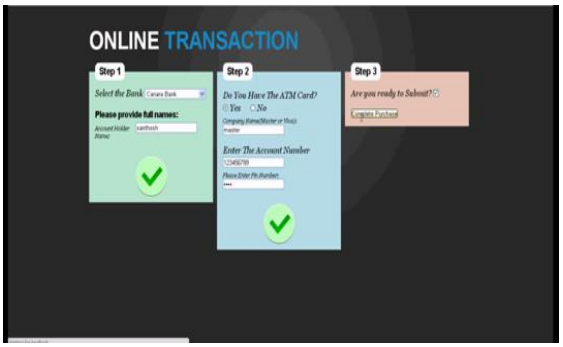
F-service should be accompanied with a special T-service so that test executions of the F-service can be performed by the consequent T-service. Thus, the normal operation of the original F-service is not disturbed by test requests and the cost of testing are not charged as real invocations of the F-service. The F-service provider can differentiate real requests from

the test requests so that no real world effect is caused by test requests. F-service should also provide additional support to other test activities.

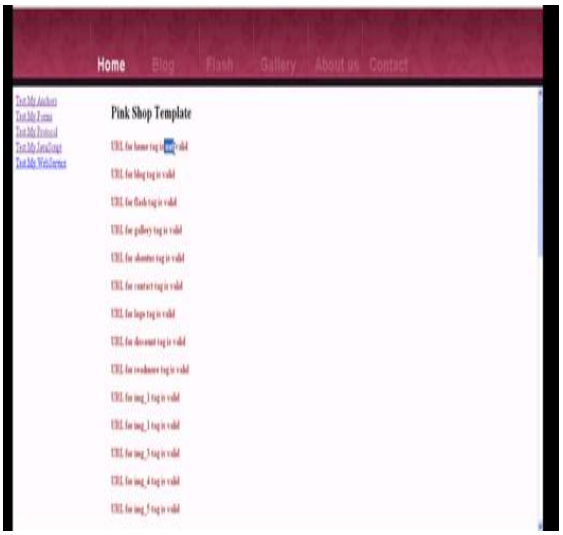


4.2Test case generation:

Besides the service specific T-service that accompanies an F-service, a test service can also be a general function test tool that performs various test activities, such as test planning, test case generation, and test result checking, etc. A general purpose T-service can be focused in certain testing techniques or methods such as the generation of test cases from WSDL. The test broker TB decomposes the test task into a sequence of subtasks and searches for appropriate testers for each subtask by submitting search requests to the registry.



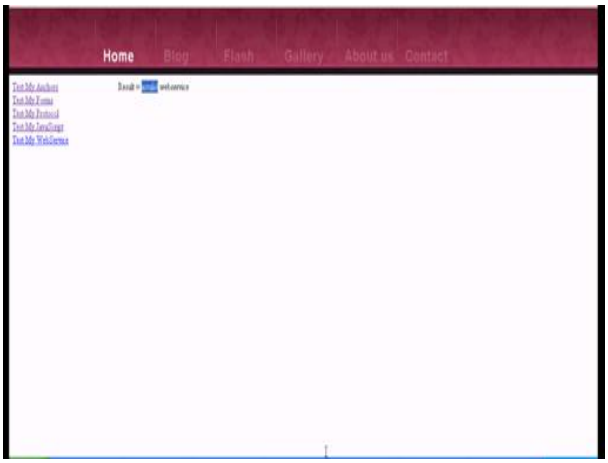
4.3Result Checking:



After checking the trustworthy of tester TG, the insurer A's T-service releases its design model to TG. After effectively obtain the design model, TG produce a set of test cases and returns a test suite to the test broker TB. The test broker then pass the test cases to TE, requests for the test summons of the insurer A's services using the test cases and requests it to check the output correctness and to measure the test coverage. TE performs these tasks by association with the insurer A's T-services. The test results are then returned to the test broker TB.

4.4 Report Preparation:

Finally, TB assembles a test report containing information about test output correctness and test sufficiency. The test report is sent to TB, which is used to decide whether the dynamic link will take place.



V. Conclusion

we presented a service oriented architecture for testing Web Services. In this architecture, various T-services collaborate with each other to complete the test tasks. We employ the ontology of software testing STOWS to describe the capabilities of T-services and test tasks for the discovery, registration and invocation of T-services. The knowledge of intensive composition of T-services has realized by the development and employment of the test brokers, which are also called T-services. We have implemented the architecture in Semantic WS technology. Here Errors can be identified in particular location. In background side web services to display fault links. By using JUnit tool we can checking the each and every communication links.

References

[1] F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, Web Services Architecture, W3C Working Group Note, http://www.w3.org/TR/ws-arch,2004.
[2] L.F. de Almeida and S.R. Vergilio, “Exploring Perturbation Based Testing for Web Services,” Proc. IEEE Int’l Conf. Web Services (ICWS ’06), pp. 717-726, Sept. 2006.
[3] J.B. Li and J. Miller, “Testing the Semantics of W3C XML Schema,”Proc. 29th Ann. Int’l Computer Software and Applications Conf. (COMPSAC ’05), pp. 443-448, July 2005.
[4] W. Tsai, R. Paul, W. Song, and Z. Cao, “Coyote: An XML-Based Framework for Web Services Testing,” Proc. IEEE Int’l Symp. High Assurance Systems Eng. (HASE ’02), pp. 173-174, Oct. 2002.
[5] X. Bai, W. Dong, W. Tsai, and Y. Chen, “WSDL-Based AutomaticTest Case Generation for Web Services Testing,” Proc. IEEE Int’l Workshop Service Oriented System Eng. (SOSE ’05), pp. 215-220, Oct.2005.
[6] N. Looker, M. Munro, and J. Xu, “WS-FIT: A Tool for Dependability Analysis of Web Services,” Proc. 28th Ann. Int’l Computer Software and Applications Conf. (COMPSAC ’04), pp. 120-123, Sept.2004.
[7] A. Bertolino, J. Gao, and E. Marchetti, “XML Every-FlavorTesting,” Proc. Second Int’l Conf. Web Information Systems and Technologies (WEBIST ’06), pp. 268-273, Apr. 2006.
[8] W. Xu, J. Offutt, and J. Luo, “Testing Web Services by XMLPerturbation,” Proc. IEEE 16th Int’l Symp. Software Reliability Eng.(ISSRE ’05), pp. 257-266, Nov. 2005.
[9] S.C. Lee and J. Offutt, “Generating Test Cases for XML-Based Web Component Interactions Using Mutation Analysis,” Proc. 12th Int’l Symp. Software Reliability Eng. (ISSRE ’01), pp. 200-209, Nov. 2001.