BLACKBOX TESTING OF WEB SERVICE

KAUSHIK RANA

Computer Engineering Department, Government Engineering College Modasa, Gujarat, India,
kaushik.rana@gecmodasa.org

Follow this and additional works at: https://www.interscience.in/ijcsi

Part of the Computer Engineering Commons, Information Security Commons, and the Systems and Communications Commons

Recommended Citation

DOI: 10.47893/IJCSI.2014.1178
Available at: https://www.interscience.in/ijcsi/vol4/iss2/2

This Article is brought to you for free and open access by the Interscience Journals at Interscience Research Network. It has been accepted for inclusion in International Journal of Computer Science and Informatics by an authorized editor of Interscience Research Network. For more information, please contact sritampatnaik@gmail.com.
BLACKBOX TESTING OF WEB SERVICE

KAUSHIK RANA

Computer Engineering Department, Government Engineering College Modasa, Gujarat, India
E-mail: kaushik.rana@gecmodasa.org

Abstract- Web Services and Service Oriented Computing (SOC) paradigm have received significant attention due to its wide spread adoption and promotion by major IT vendors. As more and more Service-Oriented Softwares are built today testing of web service have becomes crucial point. Web services are distributed, loosely coupled, autonomous, reusable, discoverable, hides internal logic (Abstraction), minimize retaining information (Statelessness) and adhere to Service Level Agreement (SLA). These inherent characteristics impose great challenges to the tester. In this paper we try to impose hierarchical structure on Web Service Description Language (WSDL) in order to uncover more and more errors before its final deployment through blackbox testing. Our approach differs from others by imposing hierarchical structure on WSDL, finding various dependencies like data dependency, control dependency, and generation of Web Service Dependence Graph (WSDG) which are essential for testing.

Keywords- Concerns, Scattered, Tangled, Object, Fan in.

1. INTRODUCTION

Web services use standard protocols such as Web Service Description Language (WSDL) to describe service description, Universal Description Discovery and Integration (UDDI) to publish WSDL, and Simple Object Access Protocol (SOAP) to exchange messages among services. A typical web service call includes interaction among the three components: service provider, service registry and service consumer as shown in Figure 1. The service provider design WSDL document and publishes in the service registry (UDDI). The service consumer looks up the service registry (UDDI) for possible match against business needs, finds the service endpoint (URL addresses), and finally calls the matched web service by exchanging SOAP messages. [1] have put a first step towards WSDL based testing or more concretely blackbox testing.

2. RELATED WORK

The complex nature of Service Oriented Software (SOS) and speedy deployment of web services inevitably leads to oversights. Testing web services reduces errors, increases the user’s confidence, and hence increases quality of SOS. Testing of web services spans through all these layered protocols. Standardized protocols leads to efficient interoperability among heterogeneous systems. WSDL enables loose coupling among web services. It contains abstract description (PortType, operation, message) and concrete description (binding, port, service). However, these informations are not sufficient to test web services, one also requires data dependency and control dependency. Testing web service focusing on WSDL description enables us blackbox testing, where service code is unavailable. However one should mention that even after carrying out thorough testing it is impossible to guarantee that the software is error free [2]. A few effort have been made by researchers for testing web service based on WSDL descriptions. For example, [1] have proposed extension to WSDL description to find out various dependencies such as input-output dependency, invocation sequences, hierarchical functional descriptions, and concurrent sequence specification. [3] analyze key challenges to test web services and propose service oriented testing framework to test web service, where T-service are the testing service working on behalf of the customer and test broker services which searches and invokes testers capable of performing testing on demand of the customers. [4] considers applicability of Testing and Test Control Notation TTCN-3 for functional and load testing of web services. [5] proposes context-aware framework for testing service orchestration and choreography in the presence of context (runtime environment). [6] have illustrated various examples for whitebox, blackbox and graybox testing of services. Our approach differs from [1] by imposing hierarchical structure on WSDL through XML schema, getting information such as data dependency and control dependency to get coverage information for the test cases.

Figure 1. Web service components
3. BLACKBOX TESTING OF WEB SERVICE

To understand the graybox testing of web service, first we need to understand how a web service is created and consumed.

3.1. A typical example

Suppose there is a fictional requirement to create a web service which computes greatest common divisor of two input values of type integer and returns integer as result. To fulfill this requirements first we need to create an XML Schema Definition (XSD) file. We have developed XSD file using NetBeans IDE plugged with SOA and XML as shown in Figure 2.

The XSD file defines two complex types named as compute gcd and compute gcdResponse. The compute gcdtype have two elements arg0 and arg1 of xs:int type and the compute gcdResponse have an element return of type xs:int, which are equivalent to integer data type. Then we have created two elements namely compute gcd and compute gcdResponse of type tncompute gcd and tncompute gcdResponse respectively.

Then we create a WSDL file shown in Figure 3 conforming to the schema defined earlier. It includes the XSD file with attribute schemalocation and declare elements like PortType, Operation, Messages, Port, Binding and Service. These elements represent service/object, method, argument(s), endpoint, transport protocol and set of endpoints respectively.

Next we create a web service(compute gcd) as shown in Figure 4. The @WebService() annotation describe the class CGCD as web service. The WSDL file should be published in the service registry (UDDI), since NetBeans IDE does not support UDDI, the service client must acquire it. Then it communicates with web service by exchanging SOAP messages. The numbers on the left side indicates statement numbers (nodes). In order to test web service before deployment, the only thing available to the tester is WSDL. WSDL contains abstract information which are insufficient to test. To test web service based on WSDL a tester need to know dependencies like control and data, through which tester can check web service for possible errors. In the next section we have explained our extension to WSDL to get web service tested.

3.2. Extention to WSDL

A web service may contain number of classes, and these classes may in turn contain number of methods called as web method. The web method may contains data and/or control dependency statements. The control dependency statements are of sequence, selection and iteration types [2].

In order to get data and control dependency from WSDL, we re-create an XSD file as shown in Figure 5, forcing the service provider to adhere it. This XSD file impose hierarchical structure on WSDL and gets concrete information for testing. Figure 5 defines a WSDG as global element of complex type wsdg. The wsdg contains element class of complex type Class and an attribute Version of type Integer. And the Class encloses element method of complex type Method and attributes Name, Visibility of type string and visibility respectively. The Method encloses two elements cfg and dig of complex types CFG and DFS in sequence respectively. It also contains attributes like Name, Argument and Visibility of types string argument and visibility respectively. The CFG covers three elements Seq, Selection, Iteration of complex types seq, selection and iteration in sequence respectively. The DFS Seq, selection and iteration covers elament statement of type Integer.

Next we perform XSD validation for wellformed rules with Netbeans. We included this XSD file in WSDL using schemalocation attribute to point to its location URI as shown in Figure 6. The rounded rectangle portion in Figure 6 highlights our extension to WSDL through XSD. It defines most element WSDG and cfg statements in the form of loops (iteration), choices (Selection). Every service developer must confirm XSD and WSDL as shown in Figures 5 and 6 and publish it in order to get services to be tested.
3.3. Testing of WSDL

As shown in Figure 7 when the web service gets registered in UDDI it is being parsed by DOM (Document Object Model) parser and generates intermediate representation which we called Web Service Dependency Graph shown in Figure 8. The tester (service provider/service consumer) inputs test cases \((3,3),(2,3),(3,2)\) to compute gcd. Each statement node in a web service is probed to determine whether it gets executed or not. If a statement node is executed it is being reflected along with its paths as shown in Figure 9 for each of the test case. The output from compute gcd enables the graph to be traversed according to the test cases, enabling the tester to find possible errors.

4. CONCLUSION

In this paper we have successfully applied blackbox testing strategy to web service. The novelty of our approach are imposing hierarchical structure on WSDL, getting dependency information in the form of WSDG. We have implemented a toy implementation of our approach. Our testing process can further be improved by mixing it with specialized whitebox testing strategy like program slicing.

REFERENCES


Figure 5. Extended XSD for compute gcd
Blackbox Testing of Web Service

Figure 7. Testing of WSDL

Figure 6. Extended WSDL for compute.gcd

Figure 8. WSDG of compute.gcd

Figure 9. Test case execution on WSDG