International Journal of Computer Science and Informatics

Volume 4 | Issue 2 Article 2

October 2014

BLACKBOX TESTING OF WEB SERVICE

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

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 to-day testing of web service have becomes crucial point. Web services are distributed, loosly coupled, autonomous, reusable, discoverable, hides internal logic (Abstraction),minimize retaining information (Statelessness) and adhere to Service Level Aggrement (SLA).These inherent characterestics imposes 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 differes 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.*

I. INTRODUCTION

Web services uses standard protocols such as Web Service Descripton 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, in- creases 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 inter- operability 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 focussing on WSDL description enables us blackbox testing, where service code is unavailable.However one should mention that even after carrying out thorughrough testing it is impossible to gurantee that the software is error free [2]. A few effort have been made by reserchers for test- ing web service based on WSDL descriptions. For exam- ple , [1] have proposed extension to WSDL desrciption to find out various dependencies such as input-output depen- dency, 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 thecustomer 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 envirionment).[6] have illustrated various examples for whitebox,blackbox and graybox testing of services. Our approach differs from [1] by imposing hierachical structure on WSDL through XML schema, getting information such as data dependency and control dependency to get coverage information for the test cases.

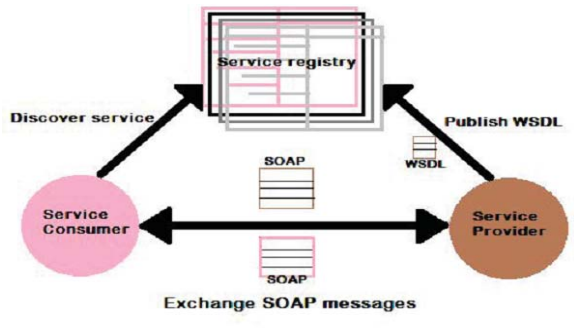

Figure 1. Web service components

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.

```
<?xml version="1.0" encoding="UTF-8"?>
- <!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net.
  RI's version is JAX-WS RI 2.1.3.1-hudson-417-SNAPSHOT.
  -->
<xs:schema xmlns:tns="http://mypackage/"
   xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0"
   targetNamespace="http://mypackage/">
<xs:element name="compute_gcd" type="tns:compute_gcd" />
<xs:element name="compute_gcdResponse" type="tns:compute_gcdResponse" />
<xs:complexType name="compute_gcd">
<xs:sequence>
<xs:element name="arg0" type="xs:int" />
<xs:element name="arg1" type="xs:int" />
   </xs:sequence>
   </xs:complexType>
<xs:complexType name="compute_gcdResponse">
<xs:sequence>
<xs:element name="return" type="xs:int" />
   </xs:sequence>
   </xs:complexType>
   </xs:schema>
```

Figure 2. XSD for compute_gcd

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 tns:compute_gcd and tns:compute_gcdResponse respectively.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--
  Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.1-hudson-417
   SNAPSHOT.
   -->
- <!--
  Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.1-hudson-417
   SNAPSHOT.
   -->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
   1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://mypackage/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   targetNamespace="http://mypackage/" name="CGCDService">
<types>
<xsd:schema>
<xsd:import namespace="http://mypackage/"
    schemaLocation="http://localhost:9529/CGCDwebservice/CGCDService?xsd=1" />
   </xsd:schema>
   </types>
<message name="compute_gcd">
<part name="parameters" element="tns:compute_gcd" />
   </message>
<message name="compute_gcdResponse">
<part name="parameters" element="tns:compute_gcdResponse" />
   </message>
<portType name="CGCD">
<operation name="compute_gcd">
   </portType>
<binding name="CGCDPortBinding" type="tns:CGCD">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
<operation name="compute_gcd">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
   </input>
<output>
<soap:body use="literal" />
   </output>
   </operation>
   </binding>
<service name="CGCDService">
<port name="CGCDPort" binding="tns:CGCDPortBinding">
<soap:address location="http://localhost:9529/CGCDwebservice/CGCDService" />
   </port>
   </service>
   </definitions>
```

Figure 3. WSDL for compute_gcd

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 represents 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()* annonation 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 extention to WSDL to get web service tested.

```
package mypackage;

import javax.jws.WebService;

/**
 *
 * @author KKR
 */
@WebService()
public class CGCD {
    public int compute_gcd(int x, int y){
1   while(x!=y){
2       if(x>y)
3           x=x-y;
4       else
5           y=y-x;
6   }
7   return x;
    }
}
```

Figure 4. Web service for compute_gcd

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 imposes 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 Cname,CVisibility of type *string* and *visibility* respectively.The *Method* encloses two elements cfg and dfg of complex types *CFG* and *DFG* in sequence respectively.It also contains attributes like Mname,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 *DFG*,*seq*,*selection* and *iteration* covers element statement of type integer.

Next we perfom 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 root element WSDG and cfg statments in the form of loops(Iteration),choices(Selection).Every service developer must confirm XSD and WSDL as shown in Figure 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) in- puts 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

[1] W.T.Tsai,Ray Paul,Yamin Wang,Chun Fan,Dong Wangl,"Extending WSDL to Facilate Web Services Testing", Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering(HASE'02),2002.

[2] Rajib Mall, "Fundamentals of Software Engineering", PHI,New Delhi,2009.

[3] Hong Zhu,"A Framework for Service-Oriented Testing of Web Services", "Proceedings of the 30th Annual International Computer Software and Applications Conferences(COMPSAC'06)",2006.

[4] Ina Schieferdecker,Bernard Stepien,"Automated Testing of XML/SOAP based Web Services", "http://www.site.uottawa.ca/ bernard".

[5] Lijun Mei,"A context-aware orchestrating and choreographic test framework for service-oriented applications ", ICSE Companion,2009: 371-374.

[6] SOA testing tool survey, "http://soatestingresearch.blogspot.in/2008/10/soa-testing-tool-survey.html".


```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://xml.netbeans.org/schema/CGCDXmlSchema"
    xmlns:tns="http://xml.netbeans.org/schema/CGCDXmlSchema"
    elementFormDefault="qualified">
    <xsd:simpleType name="Statement">
        <xsd:restriction base="xsd:integer"/>
    </xsd:simpleType>
    <xsd:complexType name="DFG">
        <xsd:sequence>
            <xsd:element name="statement" maxOccurs="unbounded" type="tns:Statement
        </xsd:sequence>
        </xsd:complexType>
    <xsd:complexType name="seq">
        <xsd:sequence>
            <xsd:element name="statement" maxOccurs="unbounded" type="tns:Statement
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="selection">
        <xsd:sequence>
            <xsd:element name="statement" maxOccurs="unbounded" type="tns:Statement
    </xsd:complexType>
    <xsd:complexType name="iteration">
        <xsd:sequence>
            <xsd:element name="statement" maxOccurs="unbounded" type="tns:Statement
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CFG">
        <xsd:sequence>
            <xsd:element name="Seq" maxOccurs="unbounded" type="tns:seq"/>
            <xsd:element name="Selection" maxOccurs="unbounded" type="tns:selection
            <xsd:element name="Iteration" maxOccurs="unbounded" type="tns:iteration
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="argument">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="([a-z] | [A-Z] | [0-9])*"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="visibility">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="public"/>
            <xsd:enumeration value="private"/>
            <xsd:enumeration value="protected"/>
            <xsd:enumeration value="privateprotected"/>
            <xsd:enumeration value="friendly"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="Method">
        <xsd:sequence>
            <xsd:element name="cfg" maxOccurs="unbounded" type="tns:CFG"/>
            <xsd:element name="dfg" maxOccurs="unbounded" type="tns:DFG"/>
        </xsd:sequence>
        <xsd:attribute name="Mname" type="xsd:string"/>
        <xsd:attribute name="Argument" type="tns:argument"/>
        <xsd:attribute name="Visibility" type="tns:visibility"/>
    </xsd:complexType>
    <xsd:complexType name="Class">
        <xsd:sequence>
            <xsd:element name="method" maxOccurs="unbounded" type="tns:Method"/>
        </xsd:sequence>
        <xsd:attribute name="Cname" type="xsd:string"/>
        <xsd:attribute name="CVisibility" type="tns:visibility"/>
    </xsd:complexType>
    <xsd:complexType name="wsdg">
        <xsd:sequence>
            <xsd:element name="class" maxOccurs="unbounded" type="tns:Class"/>
        </xsd:sequence>
        <xsd:attribute name="Version" type="xsd:integer"/>
    </xsd:complexType>
    <xsd:element name="WSDG" type="tns:wsdg"/>
        </xsd:schema>
```

Figure 5. Extended XSD for compute_gcd

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="extendedWSDL" targetNamespace="http://j2ee.netbeans.org/wsdl/extendedWSDL"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://j2ee.netbeans.org/wsdl/extendedWSDL"
    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:ns1="http://xml.netbeans.org/schema/CGCDXmlSchema">
    <types>
        <xsd:schema targetNamespace="http://j2ee.netbeans.org/wsdl/extendedWSDL"
            xmlns:ns1="http://xml.netbeans.org/schema/CGCDXmlSchema">
            <xsd:import namespace="http://xml.netbeans.org/schema/CGCDXmlSchema"
            schemaLocation="mypackage/CGCDXmlSchema.xsd"/>
        </xsd:schema>
    </types>
    <message name="compute_gcd">
        <part name="compute_gcd" type="xsd:string"/>
    </message>
    <message name="compute_gcdResponse">
        <part name="part1" type="xsd:string"/>
    </message>
    <portType name="CGCD">
        <operation name="compute_gcd">
            <input message="tns:compute_gcd"/>
            <output message="tns:compute_gcdResponse"/>
        </operation>
    </portType>
    <binding name="CGCDPortBinding" type="tns:CGCD">
        <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="compute_gcd">
            <soap:operation/>
            <input>
                <soap:body use="literal" namespace="http://j2ee.netbeans.org/wsdl/extendedWSDL"/>
            </input>
            <output>
                <soap:body use="literal" namespace="http://j2ee.netbeans.org/wsdl/extendedWSDL"/>
            </output>
        </operation>
    </binding>
    <service name="CGCDService">
        <port name="CGCDPort" binding="tns:CGCDPortBinding">
            <soap:address location="http://localhost:${HttpDefaultPort}/CGCDService/CGCDPort"/>
        </port>
    </service>
    <WSDG Version="1.0" >
        <class Cname="CGCD" Cvisibility="public">
            <method Mname="compute_gcd" Argument="int x,int y" Visibility="public" >
                <cfg>
                    <Iteration>
                        <statement>6</statement>
                        <statement>1</statement>
                        <statement>7</statement>
                    </Iteration>
                    <Selection>
                        <statement>2</statement>
                        <statement>3</statement>
                        <statement>4</statement>
                        <statement>5</statement>
                    </Selection>
                </cfg>
            </method>
            </class>
        </WSDG>
    <plnk:partnerLinkType name="extendedWSDL">
        <!-- A partner link type is automatically generated when a new port type is added. Partner
In a BPEL process, a partner link represents the interaction between the BPEL process and a partner
A partner link type characterizes the conversational relationship between two services. The partner
        <plnk:role name="CGCDRole" portType="tns:CGCD"/>
    </plnk:partnerLinkType>
</definitions>
```

Figure 6. Extended WSDL for compute_gcd

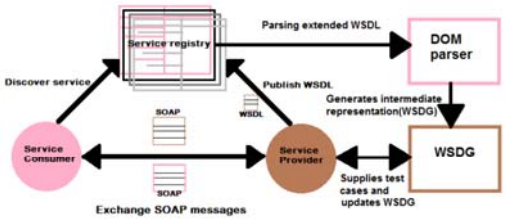

Figure 7. Testing of WSDL

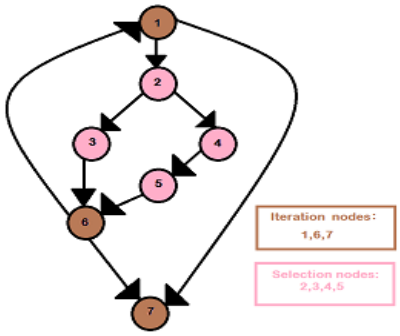

Figure 8. WSDG of compute_gcd

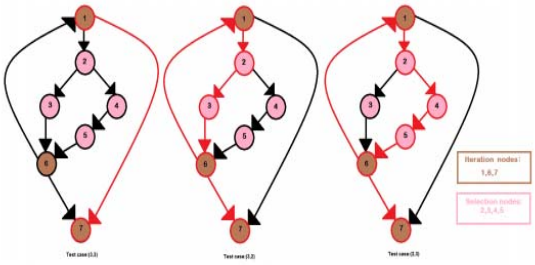

Figure 9. Test case execution on WSDG

◈◈◈