RESEARCH ARTICLE OPEN ACCESS

A Framework for Web Service Creation Consumption, Testing and Reconfiguration of Test Cases

Sirisha K L S [1], V.Santosh Kumar [2]
Associate Professor
Department of Computer Science and Engineering
Keshav Memorial Institute of Technology [1].
Sreyas Institute of Engineering and Technology [2]
Hyderabad - India

ABSTRACT

Web services technology enabled the computing world to have new phenomena in computing. Cloud computing is on top of web services only. Web service is a program that resides in server and can be invoked from across the globe by different applications in inter-operable fashion. Such web services used in application makes it a complex business application which is distributed in nature. Testing such web services is not easy. This paper provides a framework that can be used to create web services, consume them and test them. Test case reconfiguration is the important research for which a series of experiments are made. However, this paper provides partial fulfilment of automatic test case reconfiguration. It provides a portion of the framework and its implementation details. This paper is extended further in future to realize automatic test case reconfiguration when web service is subjected to different kinds of changes.

***Keywords*:-** Web service, SOA, distributed computing, test case, test case reconfiguration

I. INTRODUCTION

Web services technology is widely used in distributed computing. The distributed applications and use cases in the real world cannot be realized without this technology. Distributed applications with different services need to be tested for quality of applications. Testing web services is very important activity. However, it is tedious task as web service applications are complex and components are deployed in different servers. When web services are deployed in WWW and reused by multiple web applications, testing such applications is tedious task. It is challenging task to have test cases and test case modifications when web service is subjected to changes.

In this paper we proposed a framework for automatic discovery of web services and their changes in the source code. When changes are made, they are to be investigated to know whether new method is added, or existing method is removed, or arguments are changed. Based on the discovery of operations it needs to be analyzed. When the changes are understood, the test cases are to be reconfigured automatically. We implemented this partially in this paper and complete realization is left for our future work. We built an application with a web service. Web service client and web service test care are written. A class for web service detailed discovery based on WSDL file is defined. Thus the application is ready to discover changes in the web service periodically and notify the application.

As per the notifications, there needs to be dynamic reconfiguration of test cases. The complete functionality of dynamic test cases reconfiguration is deferred to our future work. However, in this paper we realized periodic discovery of web service changes and testing them with test cases. The remainder of the paper is structured as follows. Section 2 provides review of literature related to web services testing. Section 3 presents the proposed framework and its implementation Section 4 concludes the paper and provides directions for future work.

II. RELATED WORKS

This section throws light into the review of literature on web services and testing them. SOA based applications and their testing and test care reconfiguration is explored in [1]. Designing a broker architecture which is trust worthy and support for progressive group testing is explored in [2]. An enhanced UDDI server is used for checking and verification of web services in [3]. UDDI is used to register web services with unique service id. It supports global access to web services with standards API for discovery of web services. Web service architecture supports universal discovery and integration of web services. Java based web services testing is explored in [4] for making robust distributed applications. Robustness refers to the fact that the web services are working as per the specifications and functionalities expected. Web services can be composed in order to have distributed applications. Such compositions need to

be tested. In [5], testing of such applications is explored.

Regression tests and their safe application with web services are studied in [6]. The tests were conducted on web services built using Java language. However, they can be applicable to any web service. In case of large applications, many web services are involved. Diversified web services need to be composed correctly. Such composition is studied in [7] for more effective distributed applications. Test cases are widely used in testing. Testing web services is no exception for this. However, using ontology based approach for test case generation is investigated in [8]. The dynamics of testing web services is explored in [9] while the same with contract based approaches is investigated in [10]. Web services testing are complex and that is understood by them with test cases.

In presence of multiple web services that are part of a web based application, coverage and control flow are explored in [11] in case of testing web services. Systematic approach is desired as the composite web services are complex in nature. Web services integration over WWW is the main focus of [12] were multiple web services that run over WWW are used for web based integration and testing. The concept of passive testing of web services and their timed invariants are studied in [13]. Composite web services are studied in terms of their composition and testing in [14]. Penetration testing on web services with more attack signatures is studied in [15] for finding vulnerabilities in composite web services. In fact, testing itself is provided as service in cloud as explored in [16]. This kind of services can be used by public for testing their web service compositions.

A framework for modelling and testing web services is explored in [17]. The framework supports web service composition, workflow and testing them. In [18] test cases prioritization is investigated based on the service selection concept in location based web services. Multiple test cases with prioritized order are executed to discover flaws in application. Web service discovery is made with UDDI. However, a practical approach that can help in better way in discovering and consuming web services is presented in [19]. Test cases when used to test web series can identify bugs. However, test case reconfiguration is made manually when web services are changes in terms of functionality, arguments of new functions. This is handled by automatic test care reconfiguration as explored in [20]. In this paper, we analyze web services creation, consumption and testing with useful insights.

III. FRAME WORK FOR COMPOSING, CONSUMING AND TESTING WEB SERVICES

We proposed a framework based on the broker architecture of web services. In fact it is based on Figure 1. It shows three phases. In the first phase web service is created using Apache Axis or any such middleware for web services. Then WSDL is created for the web service. Afterwards, the web service is published in UDDI registry. The second phase is discovering web service from UDDI registry. The discovery is made using WSDL and UDDI API. Once web service is discovered, the reference is used to interact with actual web services provided by the service provider. The service requester makes the discovery process. Then service requester makes request to web service provided by service provider. This is the final phase which is nothing but the execution of web service.

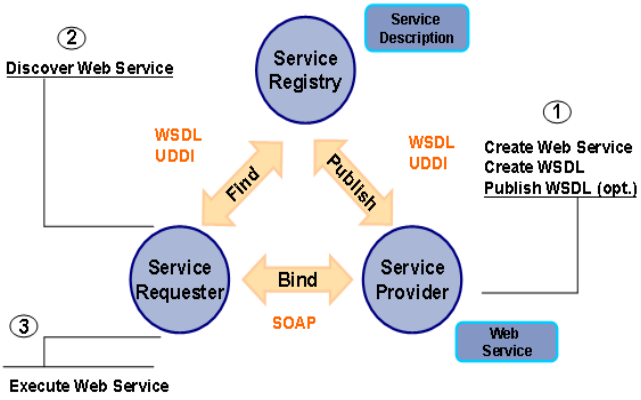

Figure 1: Web services architecture [21]

As shown in Figure 1, it is evident that web services are widely used by discovering suitable ones from UDDI registry. Service descriptions are registered in UDDI. That is the reason why web service requester needs to use WSDL for searching and locating web services. Integration of web services with different applications is based on web services technology. Web services technology leverages applications to gain access to remote services. Thus reusing existing services that are built on top of OOA is possible instead of reinventing the wheel again. We proposed and used a testing framework on top of this architecture. The rough sketch of the framework is as shown in Figure 2.

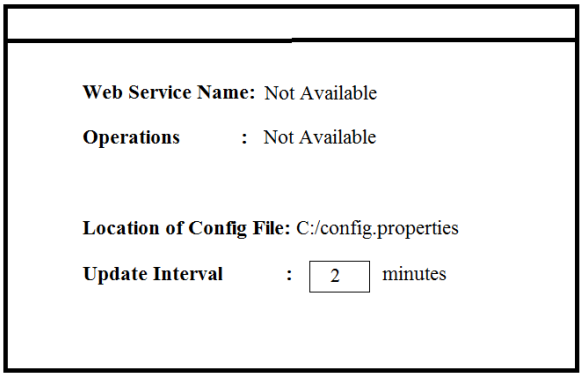

Figure 2: Part of UI for dynamic test case reconfiguration

This UI is used to find the web service name for which testing is made and the operations available. Location of configuration file provides details to the program with all information about web service. When web service is updated, all kinds of changes are taken care of by the test care reconfiguration algorithm. The complete realization of the framework is deferred to our next paper. Here parts of the framework are implemented and tested. Test cases are created and then they are automatically reconfigured when operations or arguments are changes in the target web services.

Sample Web Service

```
package com.ws.server;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

@WebService(serviceName = "CalcWebService")
public class CalcWebService {

   @WebMethod(operationName = "add")
   public int add(@WebParam(name = "arg1") int x, @WebParam(name="arg2") int y) {
       return x+y;
   }
   @WebMethod(operationName = "sub")
   public int sub(@WebParam(name = "arg1") int x, @WebParam(name="arg2") int y) {
       return x-y;
   }

 @WebMethod(operationName = "mul")
   public int mul(@WebParam(name = "arg1") int x, @WebParam(name="arg2") int y) {
       return x*y;
   }

 @WebMethod(operationName = "div")
   public int div(@WebParam(name = "arg1") int x, @WebParam(name="arg2") int y) {
       return x/y;
   }
}
```

Listing 1: Same web service

Sample Java Code for Discovering Web Service Detail from WSDL

```
public class Calculator {
 public static void main(String argv[]) throws XPathExpressionException {
   try {
        File fXmlFile = new File("C:\\Users\\sankalpa\\Desktop\\WebService\\web\\WEB-INF\\CalWebService.xml");
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(fXmlFile);
        doc.getDocumentElement().normalize();
       XPath xPath = XPathFactory.newInstance().newXPath();
        String expression =
```

```
"/definitions/operation";
        System.out.println("Root element :" +

doc.getDocumentElement().getAttribute("name"));
        NodeList nList = doc.getElementsByTagName("operation");
        System.out.println(nList.getLength());
      Node n=null;
    // Element eElement=null;
      for (int i = 0; i < nList.getLength(); i++) {
        Node nNode=nList.item(i);
       if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                Element eElement = (Element) nNode;

        System.out.println(eElement.getAttribute("name"));
        }
        System.out.println("\nCurrent Element :" + nNode.getNodeName());
      }
   } catch (Exception e) {
        e.printStackTrace();
   }
  }
}
```

Listing 2: Discovery of web service

Simple Test Case with JUnit

```
import com.ws.server.CalcWebService
import com.ws.server.CalcWebService;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;
public class CalculatorTest{
 CalcWebService c = new CalcWebService();
 @Test
 public void testAdd() {
assertEquals(22, c.add(12,10));
}
 @Test
public void test Sub() {
 assertEquals(15, c.sub(20,5));
}
 @Test
 public void test Mul() {
 assertEquals(50, c.mul(10,5));
 }
 @Test
 public void testdiv() {
 assertEquals(10, c.div(100,10));
 }
public static void main(String args[]) {
  Result result = JUnitCore.runClasses(CalculatorTest.class);
 for(Failure failure : result.getFailures()) {
 System.out.println(failure.toString());

System.out.println(result.wasSuccessful());
}
}
}
```

Listing 3: Simple JUnit Test for Testing Web Service

Setting Timer and Periodic Detection of Changes in Web Service

```
public void start() {

delay=Integer.parseInt(jTextField1.getText())*1000;
      timer.cancel();
      timer = new Timer("TaskName");
      Date executionDate = new Date(); // no params = now
      timer.scheduleAtFixedRate(task, executionDate, delay);
```

```
    }
    private class LoopTask extends TimerTask {
        public void run() {
            NewClass newclass=new NewClass();
            newclass.main(null);
        }
    }
```

Listing 4: Periodic discovery of web service changes

In connection with Figure 2, the discovery of the changes made to web service is done using Listing 4. Listing 1 provides creation of simple web service. Listing 2 shows discovery of details of web service using WSDL file. Listing 3 shows a simple test case written using JUnit. Listing 4 is used to find whether web service is subjected to changes. The changes discovered by this class are understood and test case reconfiguration is made automatically. Realization of the test care reconfiguration is not in the scope of this paper.

IV. CONCLUSIONS AND FUTURE WORK

In this paper we provide an overview of the web services creation, consumption and testing. Web services are created with broker architecture which has three phases. In the first phase, a web service is created by the service provider. Then the service is registered with UDDI registry for making it globally accessible. For doing this, the service provider creates WSDL file for providing information about web services. The WSDL is associated with registry and helps in discovery process. Service consumers or web service requester needs to have WSDL in order to make lookup and discover required web service from UDDI. Once web service reference is obtained, it is used to connect to server where web service is running actually. This is done by invoking remote web service and obtains results. Once web service is tested this way it can be integrated with applications in hand. When web services are to be tested, they can be tested with test cases. In Java we implemented web services, web services client and test cases for testing of web services. Partially we realized automatic reconfiguration of test cases when web service is subjected to different kinds of changes. In our future work to proposed and implement a comprehensive framework that supports dynamic and automatic reconfiguration of test cases.

REFERENCES

[1] M. B. Cooray, J. H. Hamlyn-Harris, and R. G. Merkel, "Test recon- figuration for service oriented applications," in Proc. IEEE Int. Conf. Utility Cloud Comput., 2011, pp. 300–305.

[2] X. Bai, Z. Cao, and Y. Chen, "Design of a trustworthy service broker and dependence-based progressive group testing," Int. J. Simul. Process Modell., vol. 3, pp. 66–79, 2007.

[3] W. T. Tsai, R. Paul, Z. Cao, L. Yu, and A. Saimi, "Verification of web services using an enhanced UDDI server," in Proc. 8th Int. Workshop Object-Oriented Real-Time Dependable Syst., 2003, pp. 131– 138.

[4] C. Barbara G. Ryder,Ana Milanova,David Wonnacott. (2004). Testing of Java Web Services for Robustness. ACM, p271-280.

[5] A. Bucchiarone, H. Melgratti, and F. Severoni, "Testing service composition," in Proc. 8th Argentine Symp. Softw. Eng., Mar del Plata, Argentina, 2007, pp. 1–16.

[6] F. Lin, Michael Ruth, Shengru Tu. (2006). Applying Safe Regression Test Selection Techniques to Java Web Services. *IEEE*, p1-10.

[7] S. Chan Oh, Dongwon Lee,Soundar R.T. Kumara. (2008). Effective Web Service Composition in Diverse and Large-Scale Service Networks. *IEEE*. 1 (1), p.20-30.

[8] Yongbo Wang, Xiaoying Bai, Juanzi Li, Ruobo Huang. (2007). Ontology-Based Test Case Generation for Testing Web Services. *IEEE*, p271-280.

[9] Reda Siblini, Nashat Mansour. (2005). Testing Web Services. *IEEE*, p1370-1381.

[10] Reiko Heckel,Marc Lohmann. (2005). Towards Contract-based Testing of Web Services. *Elsevier*. 116, p1-10.

.[11] Li Li, Wu Chou, Weiping Guo. (2008). Control Flow Analysis and Coverage Driven Testing for Web Services. *IEEE*, p271-280.

[12] Eyhab Al-Masri and Qusay H. Mahmoud. (2008). Investigating Web Services on the World Wide Web. *WWW 2008 / Refereed Track: Web Engineering - Web Service Deployment*, p.20-30.

[13] Gerardo Morales, Stephane Maag, Ana Cavalli. (2010). Timed Extended Invariants for the Passive Testing of Web Services . *IEEE*, p.20-30.

[14].Senthilanand Chandrasekaran , John A. Miller , Gregory S. Silver , Budak Arpinar & Amit P. Sheth. (2010). Performance Analysis and Simulation of Composite Web Services. IEEE, p1-14.

[15].Nuno Antunes,and Marco Vieira . (2011). Enhancing Penetration Testing with Attack Signatures and Interface Monitoring for the Detection of Injection Vulnerabilities in Web Services . IEEE, p1-8.

[16].Lian Yu, Wei-Tek Tsai1, Xiangji Chen, Linqing Liu, Yan Zhao, Liangjie Tang,and Wei Zhao2 . (2010). Testing as a Service over Cloud . IEEE, p1-8.

[17].Ana Cavalli1, Tien-Dung Cao2, Wissam Mallouli3, Eliane Martins4, Andrey Sadovykh5, Sebastien Salva6,and Fatiha Za¨idi7 . (2010). WebMov:AdedicatedframeworkforthemodellingandtestingofWebServices composition. IEEE, p1-8.

[18].Ke Zhai, Bo Jiang,W. K. Chan,and T. H. Tse. (2010). Taking Advantage of Service Selection: A Study on the Testing of Location-Based Web Services through Test Case Prioritization . IEEE, p1-8.

[19].Colin Atkinson, Philipp Bostan, Oliver Hummel and Dietmar Stoll . (2007). A Practical Approach to Web Service Discovery and Retrieval . IEEE, p1-8.

[20].Mark B. Cooray, James H. Hamlyn-Harris and Robert G. Merkel . (2013). Dynamic Test Reconfiguration for Composite Web Services . IEEE, p1-13.

[21] Web Services Architecture. Retrieved from http://computerscienceimaginarium.blogspot.in/2013/01/web-services-architecture.html