International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 59-66
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

AUTOMATIC TEST CASE GENERATION of WEB SERVICES USING CLASS DIAGRAM

Abhishek Kumar
Meerut Institute of Engineering and Technology, Meerut, India

Manindra Singh
New Delhi, India

Akhilesh Kumar Pandey
Benefitalign Technologies Pvt Ltd.
Bangalore, India

***Abstract*— Web services are formally and fully described using an XML-based document called the Web Services Description Language (WSDL) document. WSDL documents fully describe a Web service, including the operations that it supports, the messages that it exchanges, and the data types that these messages use. The best way to approach a WSDL document is to understand that different XML elements take responsibility for describing different levels of detail. This paper presents an automatic test data generation of web services through class diagram. First, class diagram of web service is designed. After that we convert this class diagram into the XML format. Further, XML instances for each class are generated. Now we apply various XSD constraints into the XML instances of classes.**

***Keywords*—** Web Service, Class diagram, XML schema, Automatic test case generation

I. INTRODUCTION

The Web service is based on open standards such as HTTP and XML-based protocols including SOAP and WSDL, Web services are hardware, programming language, and operating system independent [11]. Web services are powered by XML and three other core technologies: WSDL, SOAP, and UDDI. Before building a Web service, its developers create its definition in the form of a WSDL document that describes the service's location on the Web and the functionality the service provides. Information about the service may then be entered in a UDDI registry, which allows Web service consumers to search for and locate the services they need. Services are used over network. Messages are transmitted through interfaces in a platform neutral, standardized format, most often XML. Every service maintains a set of local state and the change of a local state depends on sending and receiving messages to and from message queues. To meet business goals several possible services combine and form another service evolution dimension where changes may not occur at a service level, but service composition structure is changed where one or more services entered or left the composition. This encourages the need to test the Web Services.

Web services are formally and fully described using an XML-based document called the Web Services Description Language (WSDL) document. WSDL documents fully describe a Web service, including the operations that it supports, the messages that it exchanges, and the data types that these messages use. The best way to approach a WSDL document is to understand that different XML elements take responsibility for describing different levels of detail. For example, the < message > element is a detailed listing of the types that factor into a given message. On the other hand, the < operation > element simply lists the messages that factor into a given operation without going into any detail as to what these messages look like [8]. Service consumers and service testers are not the developers of the web services. They can only access the web service through its interface (WSDL) without knowing the internal implementation. As a result, only black-box testing is possible for web services. Service-centric application is difficult to test due to its dynamic and adaptive nature. Two or more services combine to form a composite service to meet business goals. This service composition may change the service composition structure. So, there is a need to test the service after each evolution and need to generate the test cases that can test the new composite service. Test cases are not only used for detecting the faults in system behavior, but are also used for proving the non-existence of known errors. So, testing is also important when there is any evolution in the service or when service needs maintenance.

Test result components describe actual response of web service under test for each of the test data in each test case. Web service implementation is the source code of the web service that is written by the web service provider [1]. If the test case includes an XML message that contains an XML element we say that XML element is covered by the test case. A test case may follow a particular sequence which may cover various services [4]. The WSDL can be extended to include two sub-elements, WSFParents and WSFChildren, which are of WSFParentType and WSFChildrenType. The WSFParents is the main description, while the WSFChildren is the sub-

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 59-66
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

description. Sequence specification shows how calling sequence of services is incorporated into the WSDL [2, 5].

This paper uses UML Class Diagram to generate the test cases of Web Services. Here, we use XML concept with the class diagram.

II. RELATED WORK

Gorthi et al. [6] proposed a concept called behavioral slicing to structure activity diagram. Use case activity diagram (UCAD) represent the functional behavioral of the given use case. UCAD is a requirement specification model that is developed during the requirement analysis phase. This approach also generate new test case including modified test case to validate changed software. This paper proposes a novel concept called behavioral slicing to provide structure to the activity diagram. When we consider behavioral slicing each use case is considered into a set of unit of behavior. Unit of behavior is a tuples consisting of user action, system processing, optional candidates and system output. Here each unit of behavior represent a user action followed by system processing and system output. This approach identifies the impact on software when there are changes made to the specification. Based on this impact a test case to be re-executed to validate the software being modified. System behavior is modeled through the UCAD. Khan et al. [7] proposed a model based regression testing approach. Model describing service interfaces before and after the changes are compared in order to analyze the system evolution and identify which test need to be rerun and where new one are required. Athira et al. [3] proposed a model based test prioritization using activity diagram and identifies the difference between original model and modified model and based on this information plot a activity paths for each test case and identifies the most promising path. The test cases that cover these path are the most beneficial test cases. Bai et al. [10] introduced the web service automation testing technique based on WSDL specification. This paper mainly focuses on the atomic service automation testing and for this operation dependency analysis is used. Here three type of dependency are identified:- Input dependency, Output dependency and input/output dependency and these dependencies are used in test generation. Web service testing is automatically based on WSDL web service specification language. WSDL contains information about the service interface, Service operation, and the data that is to be transmitted. There is multiple operation in a service. We generate a test case for testing individual operation as well as sequence of operation. Jeff Offutt and Wuzhi Xu [12] present new approach to test web service based on data perturbation. Data perturbation is a process to modify request message, resending the modified request message and analyze the response message for correct behavior. Here, data perturbation is used to test peer-to-peer interaction between service. In peer-to-peer interaction there is a message exchange between two web service. Two methods are introduced related to data perturbation: data value perturbation and interaction perturbation. This approach is applicable only between two components at a time. For example, Web service A sends a request to Web service B under HTTP over the Internet. This request be a message for an XML data document. Web service B sends a response message back to A. This response be a computation result from an XML document. Harry M. Sneed and Shihong Huang [9] develop a tool called WSDLTest for supporting web service testing. The tool generates web service requests from the WSDL schema and adjusts them in accordance with pre-condition assertion written by the tester. WSDLTest tool is based on the schema of the WSDL description. From the schema two objects are generated one is a WSDL request with random data and the other is a test script. The WSDL schema is the oracle of the WSDLTest where the test result is compared. Schema can be generated automatically from the interface design or the developer writes it manually. Zhang et al. [13] extends UML 2.0 activity diagram to describe the syntax and behaviors of BPEL. This paper map BPEL primitive activity to UML 2.0 activity diagram. In BPEL < invoke >, < receive >,< reply >,< assign >, < wait > etc. are the primitive activity. These primitive activities map action node of UML2.0 activity diagram. < Invoke > map to call operation action. < Receive > map to accept call action. < Reply > map to reply action. < Assign > evaluates the value of the variables and map to read variable action and write variable action. < Wait > accept event action. Next, mapping done from BPEL structure activity to UML 2.0 activity diagram. In BPEL,< sequence >,< switch >, < flow >, < while > etc. are the structure activity.

III. PROPOSED TEST CASE GENERATION APPROACH

To generate the test cases of web services first we generate the test data of that web service using Class Diagram. We use following three steps to generate the test data. These three steps are as follow-

Step 1: - We draw the class diagram for the given web service. Further, we convert this class diagram into an XML schema format.

Step 2: - Next, we generate XML instances from the XML schema which we got through class diagram.

Step 3: - After generating XML instances now we apply various XSD constraints on it which help in customization of test cases.

We take an example of 'Online Shopping System' web service to explain our approach. We got information about web service classes and web service operations from class diagram. Web service operations represent the functionalities of web service. Each web service operations have their own

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 59-66
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

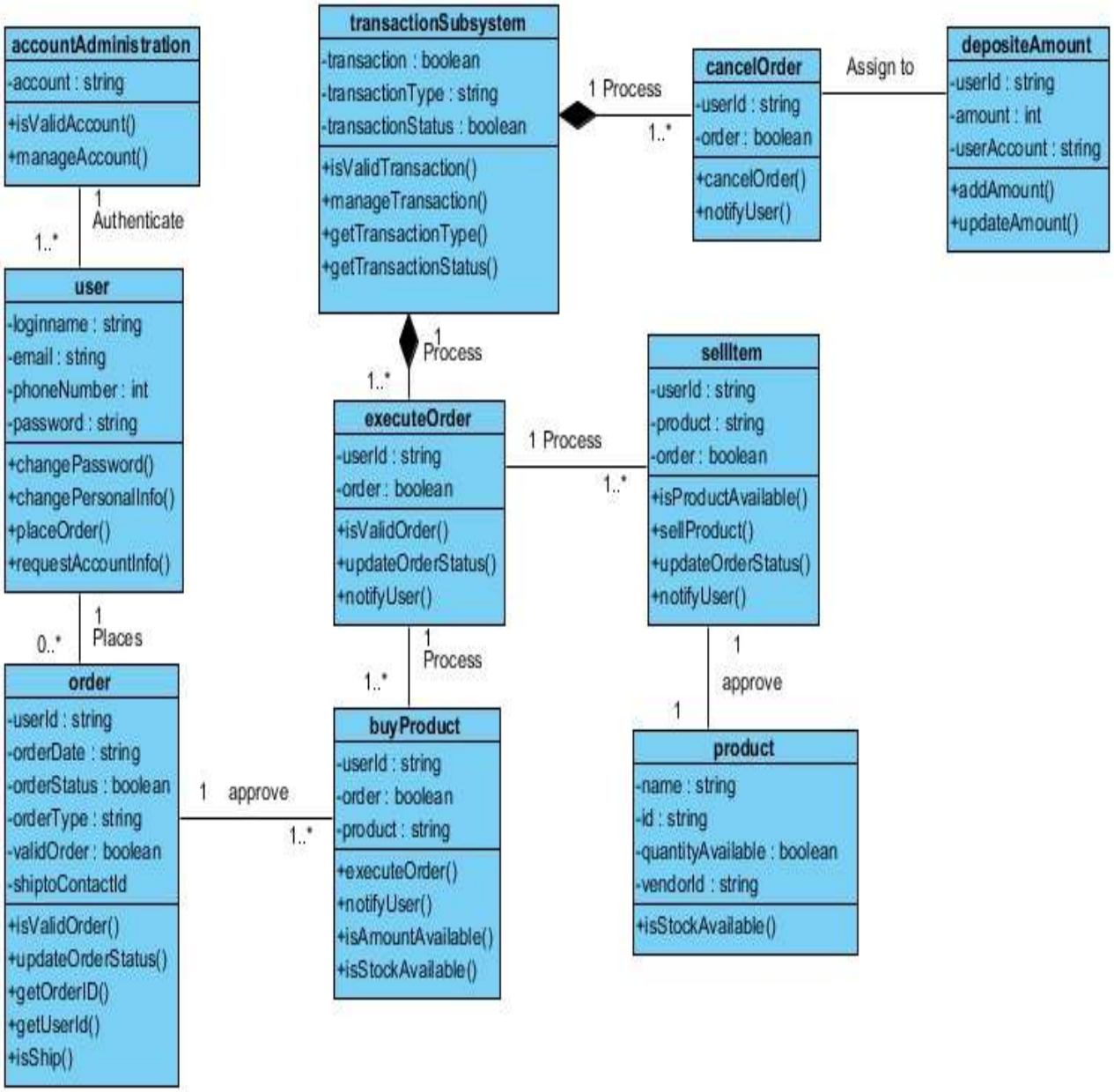

Fig 1: Class Diagram of Online Shopping System

preconditions and post conditions which help us to set a entry and exit criteria. Preconditions expressed the information about arguments passed into the service operation and the information about the system state. Post condition gives a information about the service behaviour when the service operation get executed. We use 'Visual paradigm 10.2' tool to draw a class diagram. Further, we use this tool to convert the given class diagram into the XML schema. XML code of the given class diagram as shown in Fig.1.

A **XML code of the given Class Diagram (Fig. 1)**

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 59-66
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="http://myproject" xmlns="http://myproject"
xmlns:myproject="http://myproject" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="accountAdministration">
<xs:all>
<xs:element name="account" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Authenticate" type="myproject:user" minOccurs="1" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="transactionSubsystem">
<xs:all>
<xs:element name="transaction" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="transactionType" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="transactionStatus" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element ref="myproject:executeOrder" minOccurs="1" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="user">
<xs:all>
<xs:element name="loginname" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="email" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="phoneNumber" type="xs:int" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="password" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Authenticate" type="myproject:accountAdministration" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="Places" type="myproject:order" minOccurs="0" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="order">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="orderDate" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="orderStatus" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="orderType" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="validOrder" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="shiptoContactId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
```

```
<xs:element name="Places" type="myproject:user" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="approve" type="myproject:buyProduct" minOccurs="1" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:element name="executeOrder">
<xs:complexType>
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Process" type="myproject:transactionSubsystem" minOccurs="1" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
<xs:element name="cancelOrder">
<xs:complexType>
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Process" type="myproject:transactionSubsystem" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="Assign_to" type="myproject:depositeAmount" minOccurs="0" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
<xs:complexType name="buyProduct">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="product" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element ref="myproject:executeOrder" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="approve" type="myproject:order" minOccurs="1" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="sellItem">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="product" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
```

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 59-66
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

```
<xs:element ref="myproject:executeOrder" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="approve" type="myproject:product" minOccurs="1" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="product">
<xs:all>
<xs:element name="name" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="id" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="quantityAvailable" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="vendorId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="approve" type="myproject:sellItem" minOccurs="1" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="depositeAmount">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="amount" type="xs:int" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="userAccount" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element ref="myproject:cancelOrder" minOccurs="0" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
</xs:schema>
```

Fig. 2 XML schema of the Class diagram of Online Shopping System

B. Test Case Generation

Class diagram help us to identify the various classes present in the services. After identification of the classes of the given services now we generate the XML instances for each class present in the 'Online Shopping System'. 'Online Shopping System' has many classes. For example, book order, cancel order, account administration, transaction etc. We create the XML schema of the classes. The created schema is the sub schema of the original class diagram. Communication between classes is tested through subschema generation. We generate the test data from the XML instances. This step help us to generate the test data set and so help in test case generation.

<xs:elementname="Authenticate"type="myproject:user"
minoccur="1"maxoccur="unbounded">

XML instance is created from the this subschema. XML instances help us to generate the test data. After generating the test data from all instances now we are able to build a test data set. These test data set can be used in the generated XML schema of the class diagram to further generate test cases.

maxLength, minLength, maxExclusive, minInclusive, whitespace, enumeration etc. are some *XSD constraints* which is used to generate the test data set. For example, **m*axInclusive*** specify the upper bound whereas ***minInclusive*** specify the lower bound. Space, tab and line feed is handled by *whiteSpace*.

International Journal of Engineering Applied Sciences and Technology, 2016
Vol. 1, Issue 10, ISSN No. 2455-2143, Pages 59-66
Published Online August - September 2016 in IJEAST (http://www.ijeast.com)

Instance generation of the *'account Administration class'*

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://myproject"
xmlns="http://myproject"
xmlns:myproject="http://myproject"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="accountAdministration">
<xs:all>
<xs:element name="account" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Authenticate" type="myproject:user" minOccurs="1"
maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
</xs:schema>
```

Fig. 3 XML Schema of accountAdministration class

```
<?xml version="1.0" encoding="UTF-8" ?>
<accountAdministration>
<account>32096000</account>
<authenticate> </authenticate>
</accountAdministration>
```

Fig. 4 Instance of accountAdministration class

Instance generation of the *'user' class*

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="http://myproject"
xmlns="http://myproject"
xmlns:myproject="http://myproject"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="user">
<xs:all>
<xs:element name="loginname" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:element>
<xs:element name="email" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:element>
<xs:element name="phoneNumber" type="xs:int" minOccurs="0" maxOccurs="1">
<xs:element>
<xs:element name="password" type="xs:string" minOccurs="0" maxOccurs="1">
<xs:element>
<xs:element name="Authenticate" type="myproject:accountAdministration" minOccurs="1"
maxOccurs="1">
<xs:element>
<xs:element name="Places" type="myproject:order" minOccurs="0" maxOccurs="unbounded">
<xs:element>
<xs:all>
<xs:complexType>
<xs:schema>
```

Fig. 5 Schema of user class

```
<?xml version="1.0" encoding="UTF-8" ?>
<user>
<loginname>abhi</loginname>
<email>xyz@gmail.com</email>
<phoneNumber>1234056789</phoneNumber>
<password>12@wer</password>
<Authenticate> </Authenticate>
<Place> </Place>
</user>
```

Fig. 6 Instance of user class

IV. CONCLUSION AND FUTURE SCOPE

This paper introduced a new approach to generate the test cases for web services using Class Diagram. Here, first we convert the given class diagram into an XML schema and generate the test data set. Next, we apply various XSD constraints on it which help us to generate the test cases. We got a number of test cases by changing the XSD constraints values.

Sl.no.	Element Name	Data Type	Test Data	XSD Constraints	XSD Constraints Value	Test Case
1	loginname	string	abhi	maxLength	Unbounded	Valid
2	loginname	string	amit	maxLength	15	Valid
3	loginname	string	12345	maxLength	Unbounded	Invalid
4	loginname	int	23451	maxLength	Unbounded	Invalid
5	loginname	int	abhik	maxLength	Unbounded	Invalid
6	email	string	abhi@gmail.com	maxLength	Unbounded	Valid
7	email	string	null	maxLength	Unbounded	Invalid
8	email	string	123459	maxLength	Unbounded	Invalid
9	email	int	123761	maxLength	Unbounded	Invalid
10	email	int	null	minLength	30	Invalid
11	email	string	amir@yahoo.in	minOccur	1	Valid
12	email	string	amit@yahoo.in	minOccur	0	Invalid
13	phoneNumber	int	1234567890	maxLength	10	Valid
14	phoneNumber	int	12349876	maxLength	10	Invalid
15	phoneNumber	string	2349876123	maxLength	10	Invalid
16	phoneNumber	string	asdfrgsdvt	maxLength	10	Invalid
17	password	string	123sedr	minOccur	1	valid
18	password	string	agdfta23	minOccur	1	valid
19	password	string	123sedr	maxOccur	3	valid
20	password	string	null	minOccur	1	Invalid

Table 1 - Test Case Generation of User Class

V. REFERENCES

[1] Hanna, S., and Munro, M. Fault-based web services testing. In Proceedings of Fifth International Conference on Information Technology: New Generations, ITNG. (April 2008), pp. 471-476.

[2] Tsai, W., Paul, R., Wang, Y., Fan, C., and Wang, D. Extending wsdl to facilitate web services testing. In Proceedings of 7th IEEE International Symposium on High Assurance Systems Engineering (2002), pp. 171-172.

[3] Athira B and Philip Samuel. Web Services Regression Test Case Prioritization. In Proceedings ofInternational Conference on Computer Information Systems and Industrial Management Applications, IEEE (2010).

[4] Askarunisa, A., Abirami, A. M., Punitha, K., Selvakumar, B., and Arunkumar, R. Sequence-based echniques for black-box test case prioritization for composite service testing. In Proceedings of IEEE International conference on Computational Intelligence and Computing Research (ICCIC) (Dec 2010), pp. 1-4.

[5] Xiaoying Bai, Wenli Dong, W.-T. T., and Chen, Y. Wsdl-based automatic test case generation for web services testing.
In Proceedings of IEEE International Workshop on Service-Oriented System Engineering (SOSE05) (October 2005), pp. 215-220.

[6] Ravi Prakash Gorthi, Anjaneyulu Pasala, Kailash KP Chanduka and Benny Leong. Specification-based Approach to Select Regression Test Suite to Validate Changed Software. In Proceedings of15th Asia-Pacific Software Engineering Conference, IEEE (2008).

[7] Tamim Ahmed Khan and Reiko Heckel. A Methodology for Model-Based Regression Testing of Web Services. In Proceedings of Testing: Academic and Industrial Conference - Practice and Research Techniques, IEEE (2009).

[8] Jeffrey Hasan and Mauricio Duran. Expert Service-Oriented Architec-ture in C# 2005. Apress, 2006.

[9] Sneed, H., and Huang, S. Wsdltest - a tool for testing web services. In Proceedings of Eighth IEEE International Symposium on Web Site Evolution, WSE. (Sept 2006), pp. 14-21.

[10] Xiaoying Bai and Wenli Dong. WSDL-Based Automatic Test Case Generation for Web Services Testing. In Proceedings of International Workshop on Service-Oriented System Engineering (SOSE), IEEE (2005).

[11] Erin Cavanaugh. Web services: Benefits, challenges, and a unique, visual development solution,Altova white paper, http://www.altova.com/whitepapers/.

[12] Offutt, J., and Xu, W. Generating test cases for web services using data perturbation. In Workshop on Testing, Analysis and Verification of Web Services,Boston Mass (July 2003).

[13] Guangquan, Z., Mei, R., and Jun, Z. A business process of web services testing method based on uml2.0 activity diagram. In Intelligent Information Technology Application, Workshop on (December 2007), pp. 59-65.