I.J. Modern Education and Computer Science, **2015, 1, 31-37**
Published Online January 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2015.01.05

A Novel Testing Model for SOA based Services

Abhishek Kumar
Trinity institute of technology & research, Bhopal (M.P), India
Email: abhikumar695@gmail.com

Abstract—SOA (Service-Oriented Architecture) filled the gap between software and commercial enterprise. SOA integrates multiple web services. We bear to secure the caliber of web services that gives guarantee about what network services work and their output results. There is close to work has to be performed for an automatic test case generation for SOA based services. But, full coverage of XML elements is missing. To the best of our knowledge this all works do not attempt to cover all possible elements of the XML schema presents in the WSDL file. There is also a need to apply different assertions on each service operation for generating the test cases. To overcome this problem we proposed a novel testing model for SOA based application. This new testing model helps us to get the automatic test cases of SOA based application. We build up our new test model with the aid of our proposed test case generation algorithm and test case selection algorithm. In the end, we generate the test suite execution results and find the coverage of XML schema elements present in the WSDL file.

Index Terms—Automatic Test Data Generation, Automatic Test Case Generation, SOA, Testing Model.

I. Introduction

Service-Oriented Architecture (SOA) based services are loosely coupled, discoverable, reusable, interoperable and heterogeneous in nature. SOA aligned business needs and technical solutions closely. The web service is the common technology to implement the SOA based system. The web service is technology neutral, support automated discovery and uses of services and having standards protocols. But SOA has much broader scope as compared to web services. SOA is the architectural concept which is used to acquire and integrate the services. SOA separates the service interface from its implementation, i.e, on that point is clear separation between the 'What' and the 'How'. SOA is the methodology and a governance plan and web services use this concept and established the communication between the services. In an SOA environment, web service acts as a tool that provides automation in business-to-business relationship. Thus, we can state that the web service is the realization of SOA. HTTP and SOAP are common protocols used by web services for the message Exchanging. Universal Description, Discovery and Integration (UDDI) registry is used to register and locate the web services. Application functionalities are exposed by web services through web service description language (WSDL). We have two types of web services. One is atomic services and another is composite services. Atomic services do not rely on other web services to full-fill consumer needs. For example- Temperature conversion service, weather report service, etc. are considered as an atomic service. But atomic services alone can't full-fill the consumer demands. Thus, we need a composite service. Web services include many benefits such as- reusability, modularity and interoperability. But, it also included a number of concerns and challenges. For example- When a service developer developing a web service than his main concerns is about its correctness. So, a developer takes some reasonable measure, including testing to ensure whether the web service is implemented correctly or not. Similarly, when a service consumer wants to use the web service, then quality assurance is important for him. So, it is important to test the web services and ensure the tested service is the right service for use. Service consumers and service testers are not the developers of the web services. They can exclusively access the web service through its interface (WSDL) without recognizing the internal execution. As a result, only black-box testing is possible for web services.

When there is any evolution in the service or when service needs a maintenance at that time testing play an important role. Interface information of a service can be used to rank the error- detection ability of test cases. When we talk about service Integrator perspective for testing, structure of service process information and information about the interfaces of partner services are available. The responsibility of service Integrator is to check the interactive behaviors of the services and detect the changes of partner service. Interface information is coming out through WSDL. WSDL having abstract-level description information and access information. The abstract - level description provides service functional interface. Through access information service user can access the service at the concrete end point. *Port Type, operation, message* and *type* included abstract-level descriptive information. *Port* and *binding* included access information.

Processes, binding and interfaces are the three types through which we can identify the changes in the composite service. BPEL, as the de-facto standard for service composition. The Composite service which uses BPEL is a combination of partner service and process service. Process is an interface described in the WSDL specification. Partner service interacting with the process service. Process change occurs due to the change of BPEL activities and the change of activities order. Binding change includes the changes in endpoint addresses of partner services. There are two types of interface change first is a composite service interface change and partner service interface change [5]. Web service execution, reliability depends upon web service transactions. WS-transactions involve autonomous and independent partners, multiple parties cooperation and define dependency among partner service activities. Technical failure or/and service level failure may occur during the WS-transactions. So, to ensure the correctness and accuracy of SOA based services, testing of web service transactions plays an important role. Level, feature and depth are the three dimensions of testing WS transactions. These three dimensions use the basic test concept such as test conditions, test unit and test coverage items [2]. Lack of observability of the source code, lack of control of the service, cost of testing when service is not deployed, service invocations charge on peruse basis and dynamic behavior of the services make it difficult to test service-oriented system [3].

In this paper, we proposed a novel testing model which takes a WSDL file to generate the test cases. To implement our proposed testing model we take an example of temperature conversion service [17]. The remainder of this paper is organized as follows. Section II provides details about related work. Section III describes our propose test case generation approach. In section IV we are implementing our propose approach. Section V summaries the paper and draw the conclusion and future work.

II. Related Work

Jeff Offutt and Wuzhi Xu [1] introduced Data Perturbation technique. In data perturbation technique two methods are introduced named: Data Value Perturbation and Interaction Perturbation. The Data Value Perturbation method is used only between two services. XML based test data generation approach is introduced in [4] [7] [9] [11]. In these approaches, XML complex data type is decomposed into simple data type. Further, various XSD constraints are applied into this simple data type which help in the test case generation approach. Jiang et al. [8] Proposed contract mutation testing approach to test web services. To support contract mutation testing three sources of test data is introduced which are function based test data, structure based test data and error based test data. A function based test data gained through service requirement. Structure based test data information can be gained at the time of the service implementation stage. We gained error based test data when there is any error reporting during the development process. Khan and Heckel [15] proposed a model based regression testing approach. The model describes service interfaces and identify changes before and after service evolution. In order to analyze the system evolution and identifying which test cases need to be rerun and where the new one are required khan and Heckel proposed a model. The proposed model specifies external behavior of the services and its data dependence and analysis. Athira B and Philip Samuel [13] introduced an approach to identify the original model and modified model through activity part of activity diagram. Tsai et al. [16] proposed web service group testing approach (WSGT) to test the composite service. WSGT rank the atomic service and composite service. For this, In WSGT there is a voting service oracle where each input according to majority principle is stored. These majority inputs decide rank of services. Bai and Dong [10] proposed WSDL based test case generation approach. In this approach test data generation and test operation generation are two perspectives to generate the test cases. Tsai et al. [6] Introduced testability evaluation criteria in SOA software, which serve as a reference for both service providers and developers. Chen et al. [14] introduced automatic test case generation approach based on activity diagrams. Boghdady et al. [12] introduced an XML based automatic test case generation approach using activity diagram. Here, for each XML file activity dependency table (ADT) is created. These ADT should cover all the functionalities present in the activity diagram, but in a reduced form. Further, activity dependency graph (ADG) is generated from this ADT. ADG shows all the possible test paths. Test cases derived from the model are functional test cases. These test cases have the same level of abstraction as the model creating them. Activity diagram uses all basic path coverage criteria.

III. Proposed Test Case Generation Approach

To generate the automatic test cases we proposed a novel testing model (Figure 1) to test SOA based application. This novel testing model help us to generate the automatic test cases of SOA based application. We developed our novel testing model with the help of our proposed test case generation algorithm and test case selection algorithm. Next section explained our proposed testing model.

A. Proposed Service Testing Model Explanation

This section explain our proposed testing model. In our proposed model service provider first published the service into UDDI. Service tester then send a request to service provider and get a WSDL file to start testing activity. We explain our testing model in detail as follow-

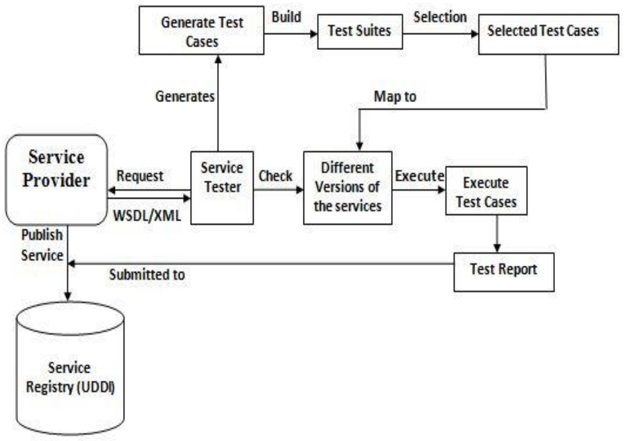


Fig.1. Proposed service testing model for SOA based application

i. **Service Provider: -** Service developer develops the web services and service provider publish this service into UDDI. Sometime service developer also known as service provider when developer himself publish the service into UDDI. Before publishing the service, service provider test the service to ensure that the service meets the service level agreement. Service provider also submits the interface information (WSDL) of the services. Service name, operator name, interface name, operation input-output parameter, operation return value, message format and location of the service information contain in the WSDL file. This WSDL file always be in XML format.

ii. **Service Tester: -** Service Tester can generates the test cases from the WSDL file. The role of tester is to generates the various XML instances of the given schema. These XML instances help us to generate the test data. After generating the test data from all the instances we have to build a test data set. This test data set can be used as input for the test cases. Service tester also check the versions of services.

iii. **Generate Test Cases: -** To generate the test cases we apply various XSD constraints into the XML schema. Some of the XSD constraints are- *maxLength*, *minLength*, *maxExclusive*, *minInclusive*, *whitespace*, *enumeration* etc. This XSD constraint helps to generate test cases. For example- *maxLength* is used to specify the maximum number of characters. Similarly, *minLength* specifies the minimum number of characters, *maxInclusive* specify the upper bound where as *minInclusive* specify the lower bound, *Whitespace* can be used to handle spaces, tab and line feed and *Enumeration* defines the acceptable value list. We generate the test cases by applying the characteristics of these XSD constraints into an XML schema. We can use test data set as an input for the generated test cases. XML schema defines the various data types. Such as: int, float, double, decimal, date, time, long, short, etc. We can change the identity of the data type to generate various ranges of the test data. For example, we can change the *string* data type into *int* to generate the *minValue* and *maxValue* of the application.

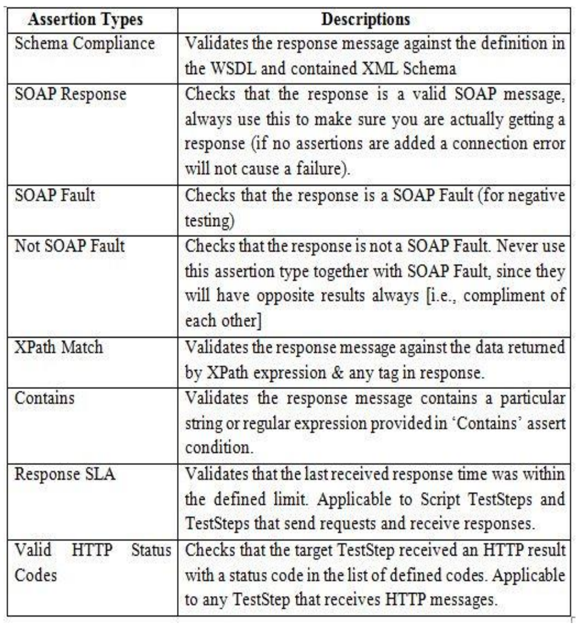
We proposed a test case generation algorithm for SOA based application. Our proposed algorithm help us to generate the test cases for SOA based services. In our algorithm precondition determines the condition under which we assign the input value. Assertion help us to decide the precondition. Assertions value used in our algorithm determine the message customization, message validation, message modification and response time validation of test cases.

B. Proposed Automatic Test Case Generation Algorithm for SOA Based Application

We proposed a test case generation algorithm which is given as follow-

1. Load the WSDL _le of service into SoapUI/ SoapUI Pro tool.
2. Identify the individual operation of a service.
3. Set the precondition by applying various assertion types. Postcondition should be the actual output value.
4. Assign the assertion value to each XML element of individual service operation.
5. Run test step.
6. if assertion value==actual output value then
7. Test pass.
8. else
9. Test fail.
10. end if
11. Repeat steps 3 to 10 until all elements of operation are traversed.
12. Exit.

To set the number of preconditions different types of assertion are used. Figure. 2 explain some assertions and their features.



Assertion Types	Descriptions
Schema Compliance	Validates the response message against the definition in the WSDL and contained XML Schema
SOAP Response	Checks that the response is a valid SOAP message, always use this to make sure you are actually getting a response (if no assertions are added a connection error will not cause a failure).
SOAP Fault	Checks that the response is a SOAP Fault (for negative testing)
Not SOAP Fault	Checks that the response is not a SOAP Fault. Never use this assertion type together with SOAP Fault, since they will have opposite results always [i.e., compliment of each other]
XPath Match	Validates the response message against the data returned by XPath expression & any tag in response.
Contains	Validates the response message contains a particular string or regular expression provided in 'Contains' assert condition.
Response SLA	Validates that the last received response time was within the defined limit. Applicable to Script TestSteps and TestSteps that send requests and receive responses.
Valid HTTP Status Codes	Checks that the target TestStep received an HTTP result with a status code in the list of defined codes. Applicable to any TestStep that receives HTTP messages.

Fig. 2. Assertion Types with Explanation

iv. **Test Suite: -** We use SoapUI/ SoapUI Pro tool to test SOA based services. In SoapUI/ SoapUI Pro tool test automation start when we create a test suite. The test suite contains the collection of test cases. When we use SoapUI/ SoapUI Pro tool then it shows that number of test cases is equal to number of service operation. Each test cases also has a number of test steps. These test steps help us to give variety in our test cases.

v. **Test Case Selection: -** There is continue evolution in the service to full-fill clients need. Service evolution make changes in the particular service. We can identify the changes in the service in the following way-

I. When new operation/operations is/are added in the WSDL.
II. When new element/elements is/are added to the XML schema.
III. Remove and/or rename the operations.

When the tester identifies the changes among two or more versions of the service, then tester find-out the missed coverage items of that service. We can select the test case set 't' from the given test suite 'T' such that t ⊆T where t= {t1,t2,t3,t4.......tn}. To identify the missed coverage item tester compares the testing result between the previous version and new one version of the service.

For example- Let S is the original service and S1 is the evolved version (added some extra features in S) of S. S1 includes some additional features that are not presents in S. Now service tester executes the test case set 't' on service S and find out the testing result. The test case set 't' includes all necessary test cases for testing the original service S. When the service S is tested now tester executes the test case set 't' on service S1. The item which are not covered by test case set 't' known as missed coverage item. When tester find out the missed coverage item, then there is a need to select the test cases other then t from the given test suite 'T' that can test the missed coverage items. Here, we propose a test case selection algorithm for testing the missed coverage items.

C. Proposed Test Case Selection Algorithm for SOA Based Application

Here, we proposed automatic test case selection algorithm for SOA based application.

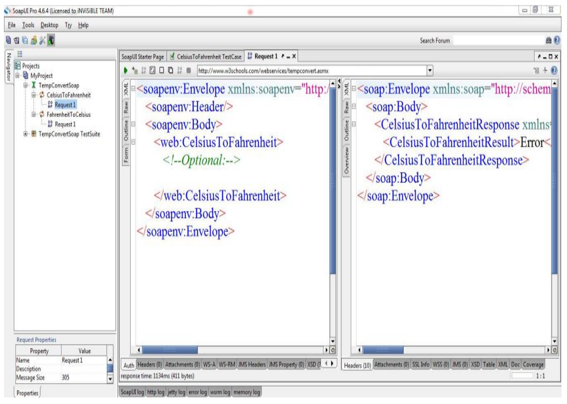
Proposed Test Case Selection Algorithm for SOA Based Application

```
1. Let T be the prioritize Test Suite and t be the Subset of T (t ⊆ T) where t={t1,t2,t3,t4.......tn}executed before and after service evolution.
2. Repeat steps 3 to 8 until all missed coverage items are tested.
3. Execute test cases other than t from T to cover all missed coverage items after service evolution. Let this test case set be T' where T'=T-t.
4. Add additional test cases say t' in T to cover missed coverage items. t' may cover additional coverage items that are not yet covered by pre-executed test cases. Here, t'={t1',t2',t3',t4'.......tn'}.
5. Check test input condition and test output condition of the test cases.
6. Execute the test cases until it cover all missed coverage items.
7. Else
8. Go to step 2
9. Exit.
```

vi. **Different Versions of the Services: -** Many different third parties are involved to provide services in the SOA environments. Each third party developed the service according to service consumer needs. Since, there are many different third parties are involved to develop the services. So, it may possible that one service may have different versions. These services are used by service consumer. When there is any evolution in the service, then it is fair to notify the service consumers about the modification of the service. But, we can't control all of our service consumers. It is also impossible to notify all the service consumers at the same time when there is any evolution in the service. In SOA environments one service has different versions and service consumers have their own choice to choose any version of that service. So, there is a need to test all the versions of the service. Test case execution result and test report should be submitted into UDDI. So that, the service consumer on request can get the detail information about the correctness of the service which they want to use.

D. Implementation of Proposed Automatic Test Case Generation Algorithm

To implement our proposed *test case generation algorithm* we use *Soap UI* tool to test our *temperature conversion service* [17]. To test the *temperature conversion service* through *Soap UI* tool first we load the WSDL location into the *Soap UI*. After adding the WSDL location into *Soap UI*, all the service operations loaded into it. In *Soap UI/ Soap UI Pro* test automation start when we generate the test suite. After generation of the test suite now we add various test steps into this test suite. These test steps help us to give variety in our test cases. Figure 3. shows the test automation when we add WSDL document into the *SOAP UI* and build test suite.


SoapUI Pro 4.6.4 (Licensed to iNViSiBLE TEAM)
File Tools Desktop Try Help
Search Forum
Navigator
Projects
MyProject
TempConvertSoap
CelsiusToFahrenheit
Request 1
FahrenheitToCelsius
Request 1
TempConvertSoap TestSuite
SoapUI Starter Page
CelsiusToFahrenheit TestCase
Request 1
XML
Raw
Outline
Form
Overview
Auth Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Property (0) XSD
Headers (10) Attachments (0) SSL Info WSS (0) JMS (0) XSD Table XML Doc Coverage
SoapUI log http log jetty log error log wsrm log memory log
Properties

http://www.w3schools.com/webservices/tempconvert.asmx

```
<soapenv:Envelope xmlns:soapenv="http:/
  <soapenv:Header/>
  <soapenv:Body>
    <web:CelsiusToFahrenheit>
      <!--Optional:-->

    </web:CelsiusToFahrenheit>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schem
  <soap:Body>
    <CelsiusToFahrenheitResponse xmlns
      <CelsiusToFahrenheitResult>Error<
    </CelsiusToFahrenheitResponse>
  </soap:Body>
</soap:Envelope>
```

Request Properties

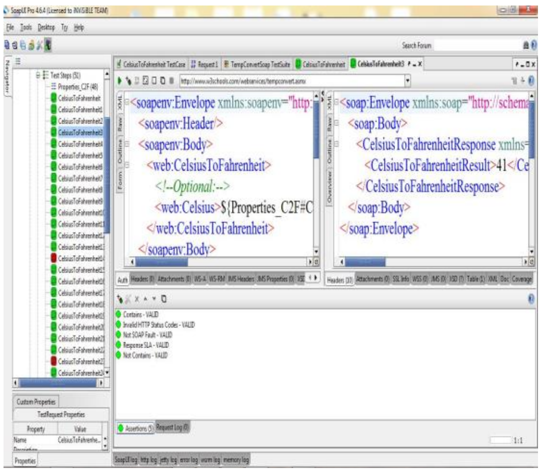
Property	Value
Name	Request 1
Description	
Message Size	305

response time: 1134ms (411 bytes)

1 : 1

Fig.3. Test Automation in SOAP UI

We can add n-number of test requests that help to assign the test data to the service operation and generate the related test response. If the test data is valid then it shows the valid test response and test cases be passed. If the test data are invalid, then testing request be filled. If applied assertion is passed, then test request be passed. If any of the applied assertions of test request be failed, then test request not passed. Figure 4 and Figure. 5 shows passed test assertion and failed test assertion. In figure 4 all assertions is passed so test request is successfully passed, but in figure 5 out of 5 assertions one assertion (Response SLA) is failing so test request is also failing. In *Soap UI/ Soap UI Pro* passed assertion shows with green color where as failed assertion shows with red color.

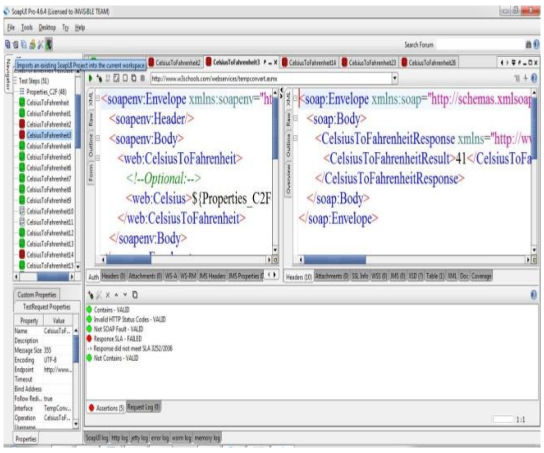


```
<soapenv:Envelope xmlns:soapenv="http:
   <soapenv:Header/>
   <soapenv:Body>
      <web:CelsiusToFahrenheit>
         <!--Optional:-->
         <web:Celsius>${Properties_C2F#C
      </web:CelsiusToFahrenheit>
   </soapenv:Body>
```

```
<soap:Envelope xmlns:soap="http://schema
   <soap:Body>
      <CelsiusToFahrenheitResponse xmlns=
         <CelsiusToFahrenheitResult>41</Ce
      </CelsiusToFahrenheitResponse>
   </soap:Body>
</soap:Envelope>
```

```
Contains - VALID
Invalid HTTP Status Codes - VALID
Not SOAP Fault - VALID
Response SLA - VALID
Not Contains - VALID
```

Fig.4. Pass assertion of temperature conversion service



```
<soapenv:Envelope xmlns:soapenv="htt
   <soapenv:Header/>
   <soapenv:Body>
      <web:CelsiusToFahrenheit>
         <!--Optional:-->
         <web:Celsius>${Properties_C2F
      </web:CelsiusToFahrenheit>
   </soapenv:Body>
```

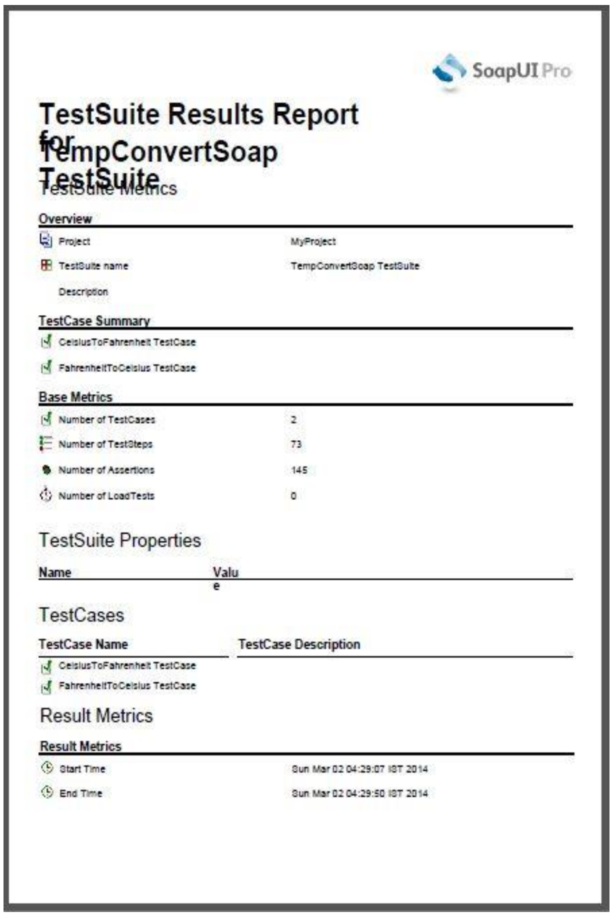
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap
   <soap:Body>
      <CelsiusToFahrenheitResponse xmlns="http://w
         <CelsiusToFahrenheitResult>41</CelsiusToFa
      </CelsiusToFahrenheitResponse>
   </soap:Body>
</soap:Envelope>
```

```
Contains - VALID
Invalid HTTP Status Codes - VALID
Not SOAP Fault - VALID
Response SLA - FAILED
-> Response did not meet SLA 3252/2036
Not Contains - VALID
```

Fig.5. Fail assertion of temperature conversion service

It is not necessary to build a separate test step to pass the test data. We can also find out the response of each test data at a single test step. But in this case last enter test data should be lost. The generation of different test steps stores the test data for the test cases under their test suite. The number of operations present in the test suite represented the total number of test cases required for that service. In our example *Temperature Conversion service*, we have two service operation named: -*CelsiusToFahrenheit* and *FahrenheitToCelsius*. However,

we can bring the variation in test cases by entering a variety of test data. We assigned the different assertion types and their value into the XML elements for each service operation. We have build the number of test steps that give variety in our test cases. So, now we execute our test suite and see the result. Here, total number of test cases is 2, because in SOA testing total number of test cases is equal to the total number of service operations. Number of test steps which give variety in the test cases are 73 and 145 numbers of assertions are used. Figure. 6 shows our test suite result report. Figure. 7 shows test coverage result that covers all the XML elements present in the individual service operation.



TestSuite Results Report for TempConvertSoap TestSuite

TestSuite Metrics

Overview

Project	MyProject
TestSuite name	TempConvertSoap TestSuite
Description	

TestCase Summary

CelsiusToFahrenheit TestCase

FahrenheitToCelsius TestCase

Base Metrics

Number of TestCases	2
Number of TestSteps	73
Number of Assertions	145
Number of LoadTests	0

TestSuite Properties

Name	Value

TestCases

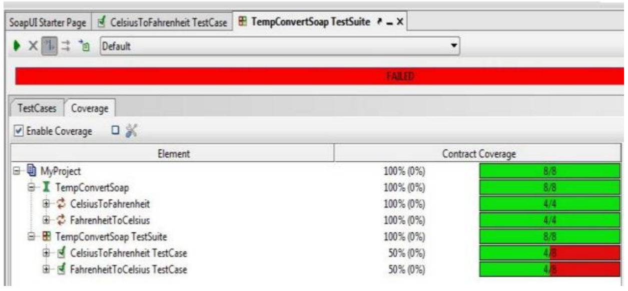
TestCase Name	TestCase Description
CelsiusToFahrenheit TestCase	
FahrenheitToCelsius TestCase	

Result Metrics

Result Metrics

Start Time	Sun Mar 02 04:29:07 IST 2014
End Time	Sun Mar 02 04:29:50 IST 2014

Fig.6. Test Suite Results Report



SoapUI Starter Page | CelsiusToFahrenheit TestCase | TempConvertSoap TestSuite

Default

FAILED

TestCases | Coverage

Enable Coverage

Element	Contract Coverage	
MyProject	100% (0%)	8/8
TempConvertSoap	100% (0%)	8/8
CelsiusToFahrenheit	100% (0%)	4/4
FahrenheitToCelsius	100% (0%)	4/4
TempConvertSoap TestSuite	100% (0%)	8/8
CelsiusToFahrenheit TestCase	50% (0%)	4/8
FahrenheitToCelsius TestCase	50% (0%)	4/8

Fig.7. Test Coverage Results

IV. Conclusions

This paper proposed a novel testing model to test SOA based services. Our proposed testing model uses the WSDL file to generate the test cases. In this paper, we discussed about test case generation technique, test case selection technique and web service versioning issue. We implemented our test case generation algorithm with the example of *Temperature Conversion Service.* Here, we apply different types of assertion of service operation elements and execute the test suite. Our proposed approach covers all the XML elements present in the individual service operation.

ACKNOWLEDGEMENT

We express our greatest gratitude and appreciation to school of computer engineering, KIIT University, Bhubaneswar, Orissa, India.

REFERENCES

[1] Jeff Offutt and Wuzhi Xu. Generating Test Cases for Web Services Using Data Perturbation. IEEE, 2003.

[2] Rubén Casado, Javier Tuya, Muhammad Younas. A Family of Test Criteria for Web Services Transactions. In the proceedings of The International Symposium on Advances in Transaction Processing. Elsevier, 2012.

[3] G. Canfora and M. Di Penta, Testing services and service-centric systems: challenges and opportunities, IT Professional 8 (2) (2006) 0–17.

[4] Hai Huang, Rick A. Mason. "Model Checking Technology for Web Services", In Proceedings of The Fourth IEEE Workshop on Software Technology for Future Embedded and Ubiquitous Systems and Second International Workshop on Collaborative Computing, Integration and Assurance (SEUS- WCCIA 06), IEEE 2006.

[5] Bixin Li, Dong Qiu, Hareton Leung and Di Wang. Automatic test case selection for regression testing of composite service based on extensible BPEL flow graph. The journal of system and software, 1300-1324, 2012. Elsevier.

[6] W. T. Tsai, Jerry Gao, Xiao Wei and Yinong Chen. Testability of Softwarein Service-Oriented Architecture. In IEEE 2006.

[7] C. Ma, C. Du, T. Zhang, F. Hu, and X. Cai. WSDL-based automated test data generation for web service. In CSSE '08: *Proceedings of the 2008 International conference on Computer Science and Software Engineering*. Wuhan, China: IEEE Computer Society, December 2008.

[8] Ying Jiang, Ying-Na Li, Shan-Shan Hou and Lu Zhang. Test Data Generation for Web Services Based on Contract Mutation. In *Proceedings of Third IEEE International Conference on Secure Software Integration and Reliability Improvement*. IEEE 2009.

[9] Z. J. Li, J. Zhu, L.-J. Zhang, and N. Mitsumori. Towards a practical and effective method for web services test case generation. In AST'09: *Proceedings of the ICSE Workshop on Automation of Software Test*. Vancouver, Canada: IEEE Computer Society, May 2009.

[10] Xiaoying Bai and Wenli Dong. WSDL-Based Automatic Test Case Generationfor Web Services Testing. In IEEE 2005.

[11] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini. WS-TAXI: A WSDL-based testing tool for web services. In ICST'09: *roceedings of the International Conference on Software Testing Verification and Validation*. Denver, Colorado, USA: IEEE Computer Society, April 2009.

[12] Pankinam N. Boghdady, Nagwa L. Badr, Mohamed A. Hashim and Mohamed F.Tolba. An Enhanced Test Case Generation Technique Based on Activity Diagrams. In IEEE (2011).

[13] Athira B and Philip Samuel. "Web Services Regression Test Case Prioritization", In *Proceedings of International Conferenceon Computer Information Systems and Industrial Management Applications* (CISIM), IEEE, 2010.

[14] Xin Chen, Nan Ye, Peng Jiang, LeiBu and Xuandong Li. Feedback-Directed Test Case Generation Based on UML Activity Diagrams. In IEEE 2011.

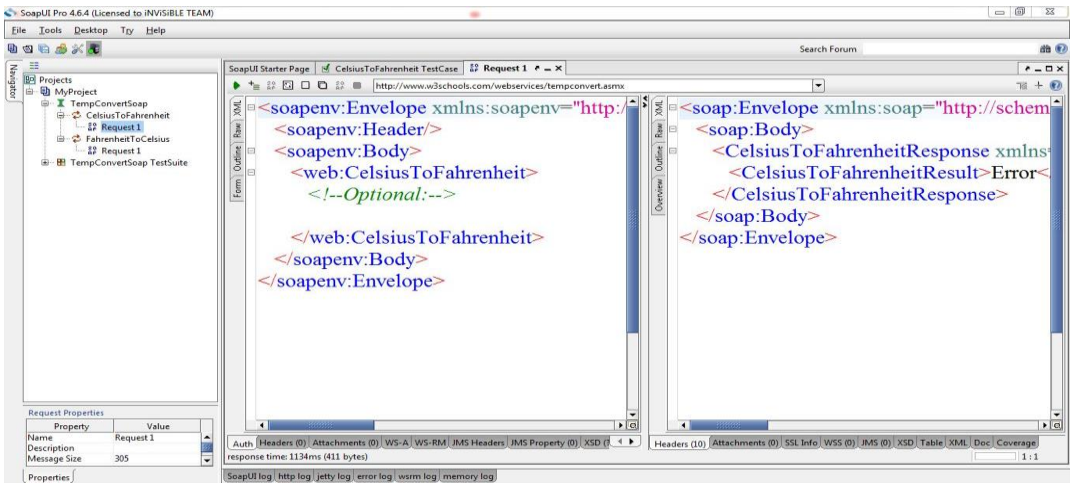
[15] Tamim Ahmed Khan, Reiko Heckel," A Methodology or Model-Based Regression Testing of Web Services", In *proceedings of Testing: Academic and Industrial Conference- Practice and Research Techniques*, IEEE 2009.

[16] W.T.Tsai, Y. Chen, R. Paul, N. Liao and H. Huang. Co-operative and Group Testing in Verification of Dynamic Composite Web Services. In IEEE (2004).

[17] http://www.w3schools.com/webservices/tempconvert.asmx?WSDL.

Authors' Profiles

Abhishek Kumar passed his M.Tech from school of computer engineering, KIIT University, Bhubaneswar, Orissa, India. He joined Trinity institute of technology & research, Bhopal (M.P) as an assistant professor in computer science department. His research area include software testing, web services and software designing.


SoapUI Pro 4.6.4 (Licensed to iNViSiBLE TEAM)
File Tools Desktop Try Help
Search Forum
Navigator
Projects
MyProject
TempConvertSoap
CelsiusToFahrenheit
Request 1
FahrenheitToCelsius
Request 1
TempConvertSoap TestSuite
SoapUI Starter Page
CelsiusToFahrenheit TestCase
Request 1
XML Raw Outline Form
XML Raw Outline Overview
Auth Headers (0) Attachments (0) WS-A WS-RM JMS Headers JMS Property (0) XSD
Headers (10) Attachments (0) SSL Info WSS (0) JMS (0) XSD Table XML Doc Coverage
SoapUI log http log jetty log error log wsrm log memory log
Properties

http://www.w3schools.com/webservices/tempconvert.asmx

```
<soapenv:Envelope xmlns:soapenv="http:/
  <soapenv:Header/>
  <soapenv:Body>
    <web:CelsiusToFahrenheit>
      <!--Optional:-->

    </web:CelsiusToFahrenheit>
  </soapenv:Body>
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schem
  <soap:Body>
    <CelsiusToFahrenheitResponse xmlns
      <CelsiusToFahrenheitResult>Error<
    </CelsiusToFahrenheitResponse>
  </soap:Body>
</soap:Envelope>
```

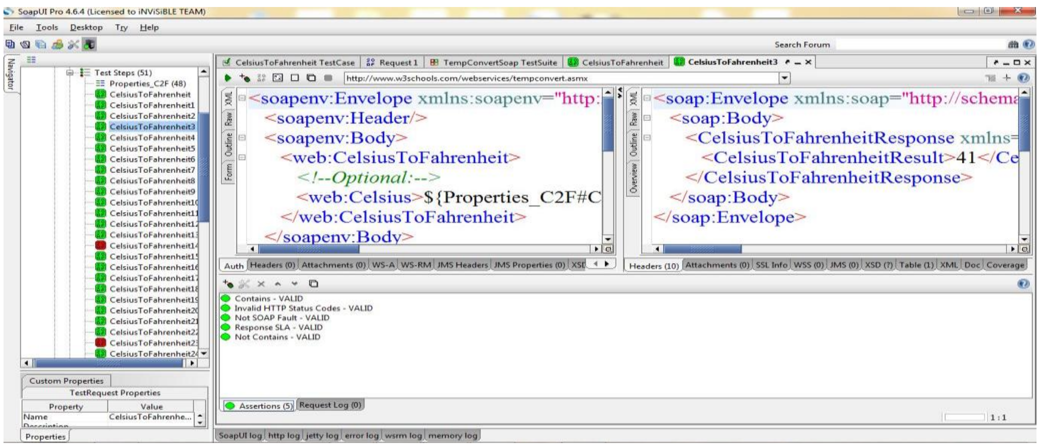
Request Properties

Property	Value
Name	Request 1
Description	
Message Size	305

response time: 1134ms (411 bytes)

1 : 1

Fig.3. Test Automation in SOAP UI

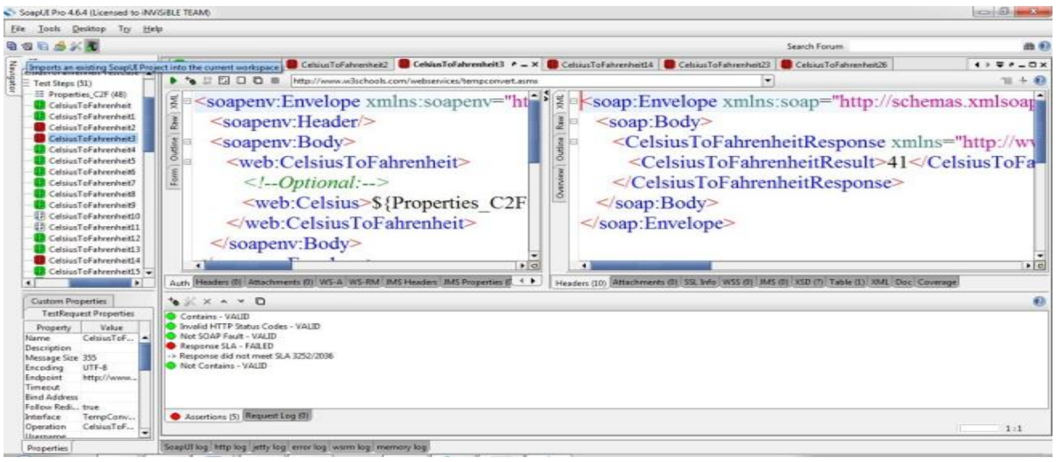


```
<soapenv:Envelope xmlns:soapenv="http:
  <soapenv:Header/>
  <soapenv:Body>
    <web:CelsiusToFahrenheit>
      <!--Optional:-->
      <web:Celsius>${Properties_C2F#C
    </web:CelsiusToFahrenheit>
  </soapenv:Body>
```

```
<soap:Envelope xmlns:soap="http://schema
  <soap:Body>
    <CelsiusToFahrenheitResponse xmlns=
      <CelsiusToFahrenheitResult>41</Ce
    </CelsiusToFahrenheitResponse>
  </soap:Body>
</soap:Envelope>
```

```
Contains - VALID
Invalid HTTP Status Codes - VALID
Not SOAP Fault - VALID
Response SLA - VALID
Not Contains - VALID
```

Fig.4. Pass assertion of temperature conversion service



```
<soapenv:Envelope xmlns:soapenv="ht
  <soapenv:Header/>
  <soapenv:Body>
    <web:CelsiusToFahrenheit>
      <!--Optional:-->
      <web:Celsius>${Properties_C2F
    </web:CelsiusToFahrenheit>
  </soapenv:Body>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap
  <soap:Body>
    <CelsiusToFahrenheitResponse xmlns="http://w
      <CelsiusToFahrenheitResult>41</CelsiusToFa
    </CelsiusToFahrenheitResponse>
  </soap:Body>
</soap:Envelope>
```

```
Contains - VALID
Invalid HTTP Status Codes - VALID
Not SOAP Fault - VALID
Response SLA - FAILED
-> Response did not meet SLA 3252/2036
Not Contains - VALID
```

Fig.5. Fail assertion of temperature conversion service

How to cite this paper: Abhishek Kumar,"A Novel Testing Model for SOA based Services", IJMECS, vol.7, no.1, pp.31-37, 2015.DOI: 10.5815/ijmecs.2015.01.05