International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 9 (2015) pp. 21749-21765

http://www.ripublication.com

A Model Based Regression Test Reduction Approach For SOA Based Applications

Rajani Kanta Mohanty[1], Binod Kumar Pattanayak[2], Durga Prasad Mohapatra[3]
[1:]Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan University, Bhubaneswar, OR, India, Email: rkm.bbs@gmail.com
[2:]Department of Computer Science and Engineering, Institute of Technical Education and Research, Siksha 'O' Anusandhan University, Bhubaneswar, OR, India, Email: binodpattanayak@soauniversity.ac.in
[3:]Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Odisha, India, Email: durga@nitrkl.ac.in

Abstract

In this paper, we present a model based test regression approach for SOA based applications in the context of web service. Here we consider an insurance system, a type of web service. The purpose behind this work is to reduce human efforts in test case generation for regression testing of a web service. We perform the tests with traditional approach as well as SOA based approach using cross platform tools. Results of tests depict that SOA based approach is more efficient in terms of spending person-days.

Keywords: SOA, WEB-SERVICES, MODEL BESED TESTING, REGRESSION TESTING

Introduction

Today's IT experts of large enterprises are adopting Service-Oriented Architecture (SOA) based application models to develop their enterprise information systems and applications. In reality, SOA based applications and web services have been used in recent times to cater the development of loosely- coupled, interoperable components and distributed applications across different platforms and hardware. Services and its application along with their underlying systems grow over time and need to be retested whenever they undergo a change, to verify that the quality has not regressed. If a small part of the system is changed, it should be possible to reuse existing tests so that the impact of changes does not need extra labour and time for the tester or developer. In this paper, we propose a model-based regression test reduction mechanism with a real life application.

Functionalities of SOA can be split into distinct units (services) [1]. Conceptually SOA is the practice of designing and developing information systems using loosely coupled interoperable software components [2]. SOA based applications offer a number of benefits and advantages such as flexibility, agility, reusability, scalability, maintainability and interoperability [3]. SOA and its more adopted implementation, Web Services, add features that need to be considered in the software development, such as distribution, lack of observability and control, and dynamic integration. In this context, the testing becomes more crucial in establishing the quality of services. The testing mechanism should be dynamic since SOA applications are constantly changing and are deployed in heterogeneous environments. Many consolidated testing approaches are applicable to the context of SOA application, though they cannot be directly termed to the dynamic and adaptive nature of services [4].

Among the existing techniques, model-based testing (MBT) is a promising candidate to provide these characteristics. MBT is an approach that derives test cases from which models are derived subsequently by the tester to describe the system under test and to support testing activity. Using MBT, the test case generation is usually efficient since the tester can update the model and regenerate the test suite, avoiding error prone manual changes [5].

In this paper we propose and compare a MBT process for SOA application versus traditional component model with respect to regression testing and establish how the time, labour and resources are saved by reduced test suites and hence the productivity and quality.

The rest of the paper is organized as follows. Section 2 covers the background. In Section 3, a traditional web-based component architecture is detailed in the context of an insurance system. The procedure for development of a web service for plug-in is discussed in Section 4. Web-based component architecture using SOA is covered in Section 5. Section 6 includes the method of deriving regression test data. Testing tools used are described in Section 7 and Section 8 concludes the paper.

Background

In this section, we present overviews of a high level architecture of web services and regression testing of the component as well as model based approaches [6].

Web Services Concept

Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network, generally, the Web. The Web Services architecture [Fig.1] is the logical evolution of object-oriented analysis and design, and the logical evolution of components geared towards the architecture, design, implementation, and deployment of e-business solutions. Web Services are deployed in different heterogonous environment and they communicate with a common language called XML which has a specific characteristics and also encapsulate all the information such as location, port ,security header ,messages in the form of SOAP envelop. Web Services Description Language (WSDL) is an XML

format for describing all the information needed to invoke and communicate with a Web Service.

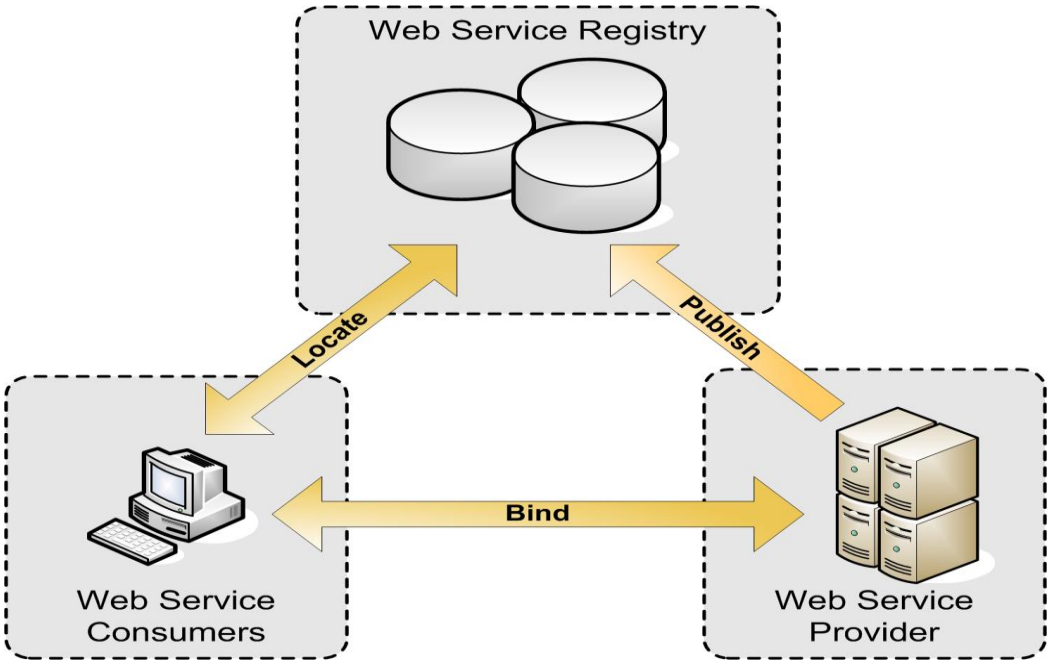

Figure 1: Web Service Architecture

After creation of web services and its deployment on a server, there are three fundamental operations of web service such as Web Service Registry, Web Service Consumer and Web Service Provider. Service provider publishes services to a service registry which acts as broker. Service consumer finds required services in the service registry, processes and sends the response as well [7].

Regression Testing Strategy

Regression means retesting the unchanged parts of the application. Test cases are re-executed in order to check whether previous functionality of application is working fine and new changes have not introduced any new bugs. This test can be performed on a newly built application, when there is significant change in original functionality or even a single bug fix. We verify that the bugs are fixed and the newly added features have not created any problem in previous working version of the software.

Regression testing is initiated when programmer fixes any bug or adds new code for introducing new functionality to the system. There can be many dependencies in newly added features and existing functionalities. It is a quality measure to check that new code complies with old code and unmodified code is not getting affected after the modification [8, 9, 10, 11, 12].

This test is very important when there are continuous changes / improvements added in the application. The new functionality should not negatively affect existing tested code.

Let us consider an insurance web application with various components which responds to the inputs as per the specification. In our study ,we will establish in the following scenarios how regression testing tends to minimize the effort using the web service call.

Scenario - 1: This scenario includes a component-based architecture that incorporates no service call and all the components are tightly coupled.

Scenario - 2: In this scenario, we present development of a web service for publisher and consumer.

Scenario - 3 : In this scenario, there is a component-based architecture that is loosely coupled with the different web services deployed in different environment.

In each scenario, we will have a different set of test cases to be executed in order to fulfill the business need as per the business requirement specifications during requirement phase which is depicted in the Section 3.

Traditional Web Based Component Architecture

In this section, we discuss the traditional web-based component architecture. We have considered the Insurance System as a case study, in order to explain the traditional web-based component architecture. In our case study, we have used the terminologies "module" and "component" interchangeably.

The various components of a general Insurance System are depicted in Fig.2. As per e-business specification document, we have designed the following architecture where various components are bound to each other and transfer messages such as Party, Policy, Account, Claim and Security , shown in Fig.2, are accessed by the end user using the valid user-id and password through security module. These modules are not SOA compatible.

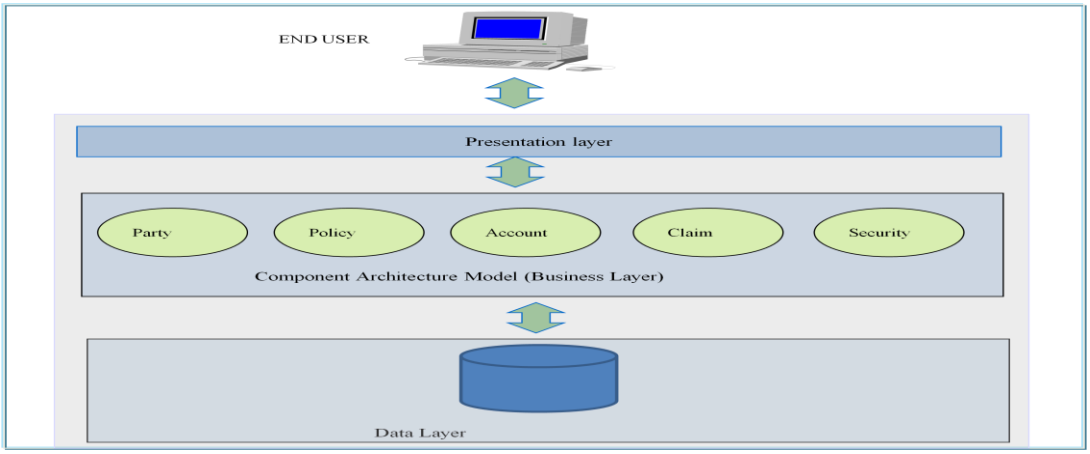

Figure 2: Components of an Insurance System

In this model, we have a two-layer abstract communication. The first layer performs the security check. The controller passes the necessary information to the

respective component where it decides about the actions to be taken for the requests. Here, the whole system is not being exposed.

Let us consider a specific component. Suppose in the component 'Party', we have defined P number of test cases for the end to end functionalities as per the business requirements. Let us define what the component 'Party' means. In the insurance industry, when a broker/agent comes to a customer for selling his policy, he has to capture the basic details of the customer such as Name, DOB, Sum Assured, Contact details, Address and so on. So here, customer is called a Party and in general called Policy Holder.

In the Party module, there may be various sub-modules such as:

Basic Details
First Name, Middle Name, Last Name, ID Type, ID Number, DOB

Primary Address Details
Address Line1, Address Line2, State, City, Pin, Country, Mobile Number, Office Number

Secondary Address Details
Address Line1, Address Line2, State, City, Pin, Country, Mobile Number, Office Number

Miscellaneous Details
PAN Number, Previous Policy Number, occupation, income etc.

During the requirement analysis, the business person has to write the test cases depending upon the above scenarios and in addition, he has to consider what dependent and independent parameters based on which the test cases are to be defined. These test cases are detailed in Table 1.

Table 1: Test Cases For Different Scenarios

Test Case No.	Field Names/ Process	Steps/Scenarios	Expected Result	Actual Result	Assigned To	Status	Remarks
TC1	Checking the input fields	Verification of optional and mandatory fields	For the mandatory input fields system should throw validation message while savng into DB.	For the mandatory input fields system should throw validation message	User1	PASS	
TC2	Checking for the Date Field	DOB Checking	DOB should be greater than 'X' years from date of inception	DOB should be greater than 'X' years from date of inception	User1	PASS	
TC3	Checking for the input field data types	Checking for the input field validation with proper data types	System should allow alphabetic characters in the name field	System should allow alphabetic characters in the name field	User1	PASS	
TCn							

These above scenarios are also dependent on Policy and Claim modules because the party will get associated with the policy as a policy holder and at the same time during the claim, policy holder will be treated as claimant. So, all the basic details and other information are required in Policy and Claim modules.

Testing Drawbacks

1. We need to find out how many dependent modules are present based on the business specification and design.
2. Used to validate software after some modifications are incorporated in the application
3. We have to not only validate the changes but also validate the functionalities due to the changes
4. Writing proper test case and test scenarios covering all impacted modules
5. Execution of test cases and verify the test results which were obtained from the previous test results.
6. Difficult to assess the functionalities derived from test results .
7. Ineffective review for the derived impacts
8. Depends on individual skill level
9. Increase in effort
10. Scalability problem when size grows
11. Difficult to update specification if there are any changes in the new requirements.

To overcome these difficulties, SOA based web services have been introduced to minimize the testing effort and other hurdles. To incorporate the SOA based web

service, we need to develop the same and call it from the component where ever required. In that case, there is no individual testing required in the other components. In the next section, we will show the web service development plug-in and how it is called from the different components.

Development of Web Service For Plug-In

In this section, we depict the Service-Oriented architecture [Fig.3] and how it is being made to plug into the above discussed insurance system. This is basically an integration of services which has been developed as web services and deployed in the server. Here, we will select one component 'Party' and establish how it can leverage the functionalities which are being called from different components. The intention of showing the detail description is to ascertain how our testing effort is getting reduced by introducing this architecture with the development and deployment of the web services.

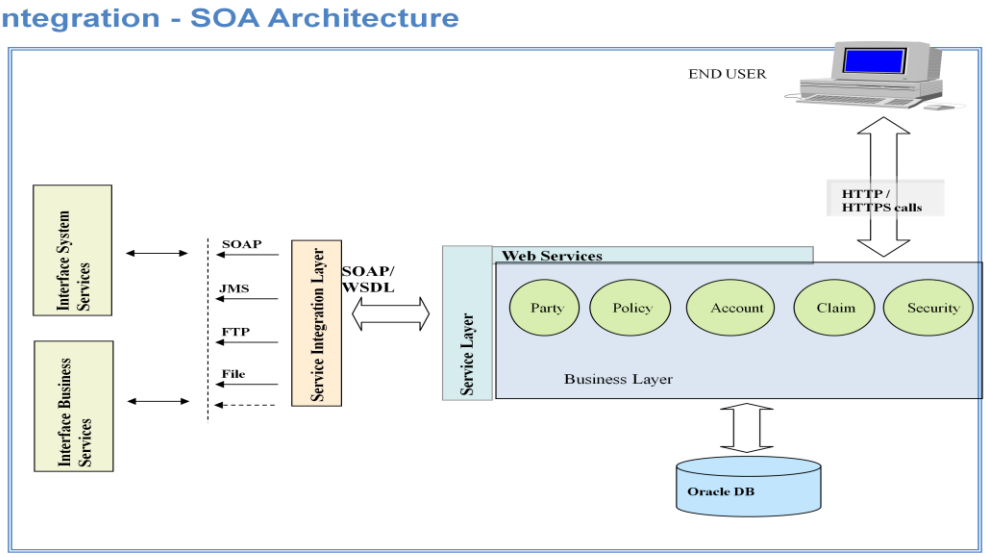

Figure 3: Integration of SOA Architecture With Cross Platform Tools

In this section, we will consider web service "PartyWS" which comprises of four different operations (method names) as follows.

Web Service Name:

1. BasicDetails_WS
2. PrimaryAddressDetails_WS (Party WS)
3. SecondaryAddressDetails_WS
4. MiscDetails_WS

Now the WSDL module for PartyWS is ready, code is ready and deployed in the server which can be accessed by the IP address and port mentioned in the WSDL.

Web Based Component Architecture Using SOA

In this section, basically we will combine sections 3 and 4 to formulate the SOA based component architecture with an intention to perform regression testing with less effort and less number of test cases.

Let us consider the same example 'Party' service of Insurance System in the SOA based architecture. We plug in our 'PartyWS' web service from Section 3 in the business layer which in turn is called from the presentation layer [Fig.2]. Now, the traditional architecture has been converted to SOA integrated architecture. In the party component, now we are having the same service being called 4 times from the different sections of the business layer. These operations can be reused from other components like Policy and Claim.

Regression Test Scenarios:

As per the business requirement, web application is always being changed from one version to another by incorporating the new changes in the different components to meet the business expectations. Modifications to one or more components might affect other components of an application, which might lead to errors. We classify modifications to web service based applications into the following types:

a) Type-1: integrating a newly established web service into the application

In this type, we merge Section 3 and Section 4 to make the web service compatible with SOA based application.

b) Type-2: adding, removing or fixing an operation in an existing component

Scenario 1:

Let us assume that we have a new business requirement from the client and after due analysis, we came up with the solution that we need to add one parameter "Email ID" in the "Primary Address details" section of the presentation layer. For that, we need to have the following steps to complete the project life cycle.

1. Modify the business specification document as per the new requirement
2. Prepare the analysis document
3. Prepare the design document
4. Coding
5. Prepare unit test cases
6. Prepare system test cases

Here the coding part is simple because of the de-coupled web service. The coder only needs to change in the PartyWS (Primary Address Details); will add one new parameter "Email ID" and necessary changes in the presentation layer to view the Email ID. Once the new parameter has been added in PartyWS, then we need to build the code and generate the WSDL. The next step is to deploy the web service PartyWS in the server. Now, our code is ready for the testing.

We have observed that there is only one change in the service and that change is one of the operations/methods (Method2). So, **the impact of the changes will be none as the operations/methods are not dependent on each other**. Hence, it implies that the coverage of the test cases will be less as compared to the traditional architecture component model. Therefore, our testing approach would be as follows:

a) Development of Web Service - Partyws Using Eclipse Tool

In this section, we have developed the web service PartyWS in which we have added new parameter 'Email ID' in the existing parameter list. Once the WSDL and underlying business logic is ready, then coding is done in the ECLIPSE and deployed in Oracle-Web logic 10.3 server [Fig.4].

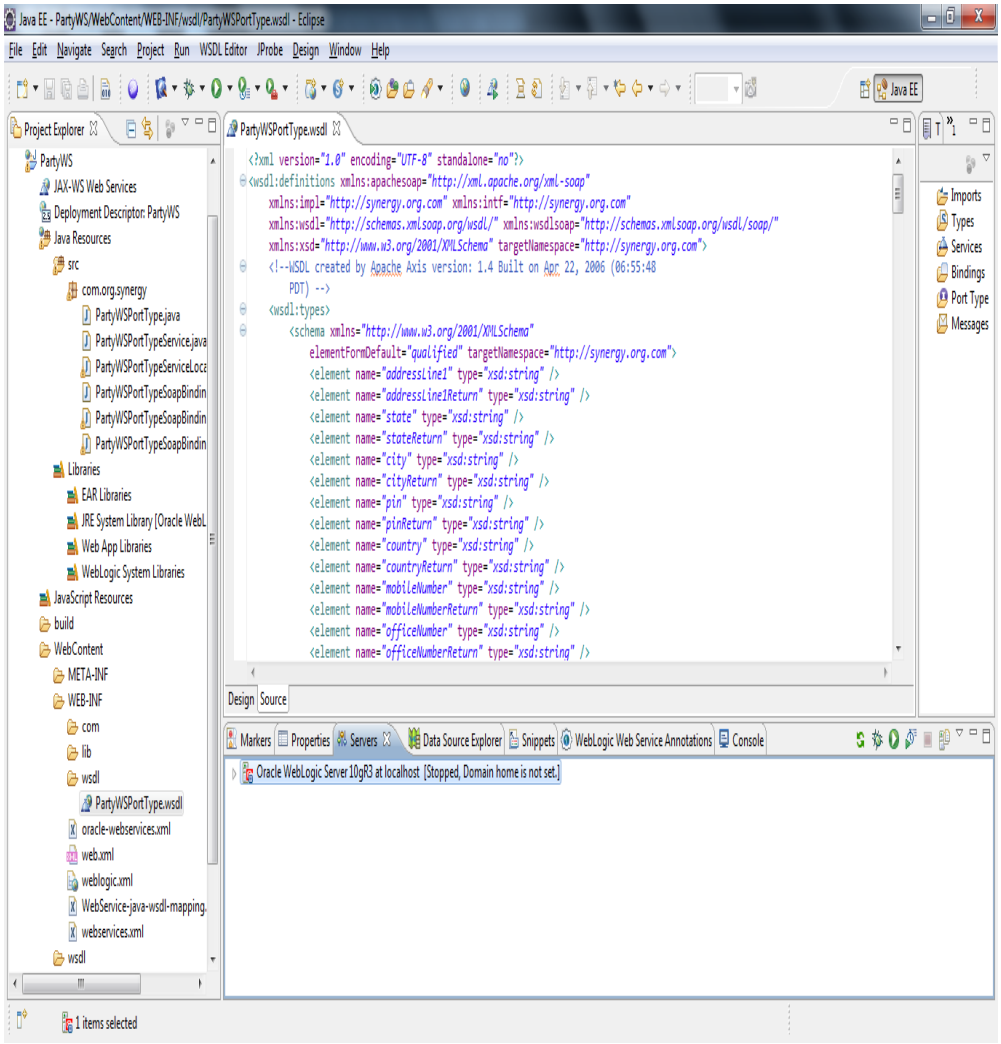

Figure 4: Snapshot of Party WS Using Eclipse Tool

b) Testing Web Service – PartyWS using SOAP UI tool

Once the PartyWS is deployed, then we test the WSDL by using the SOAP UI tool, as depicted in Fig.5.

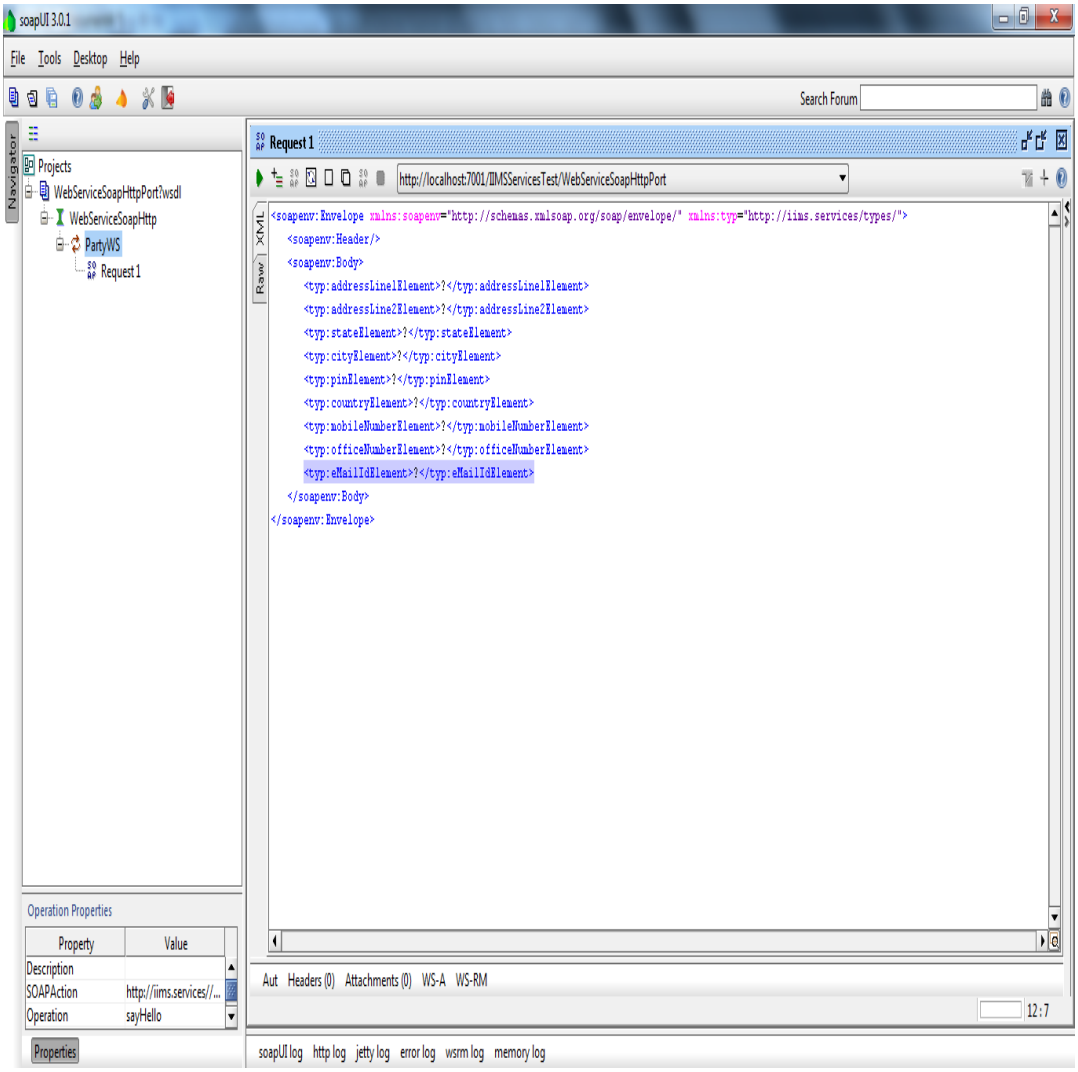

Figure 5: Snapshot of Party WS Using SOAP UI Tool

We fill up the input parameter in the SOAP UI console and press the run button. Then, we get the web service response in right hand side of the SOAP UI as success/failure [Fig.6].

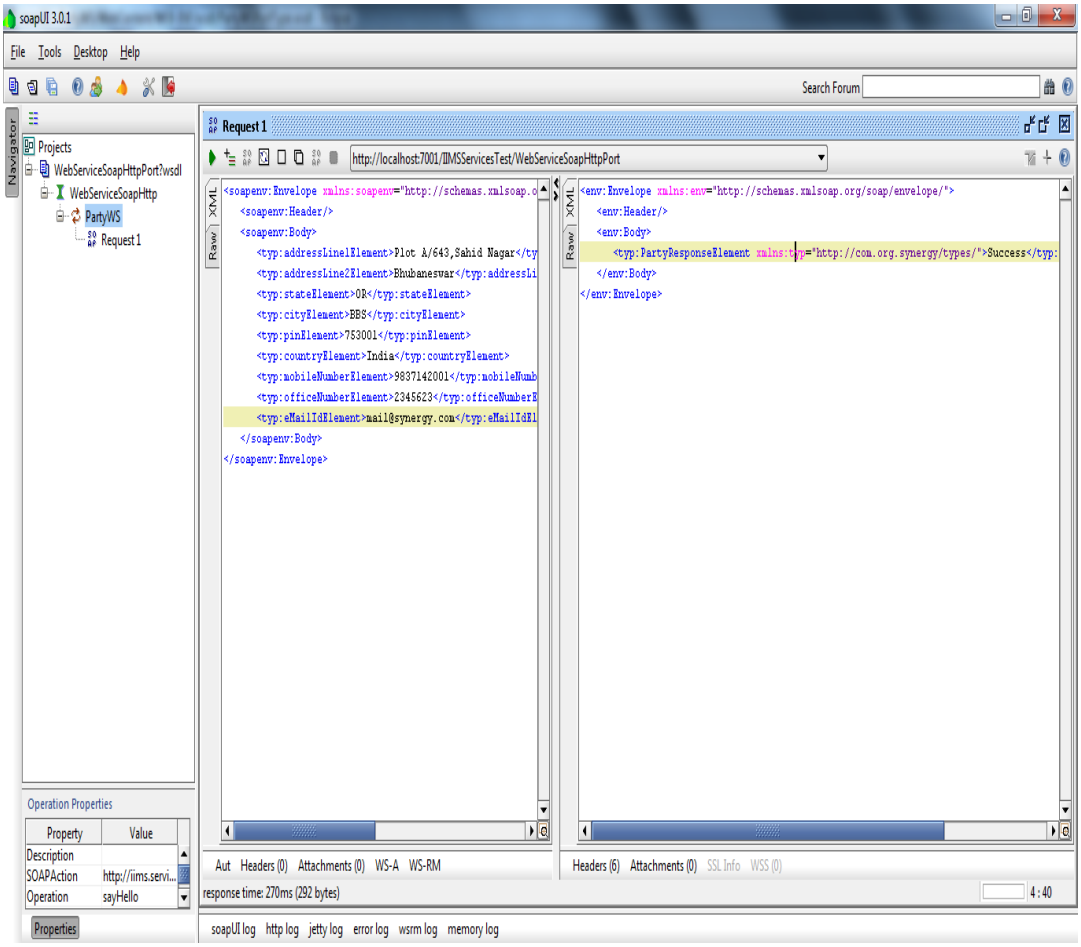

Figure 6: Snapshot of Web Service Response

If we fill up all the parameters except one mandatory parameter, then we will get the response as Failure: Mandatory parameter missing [Fig.7].

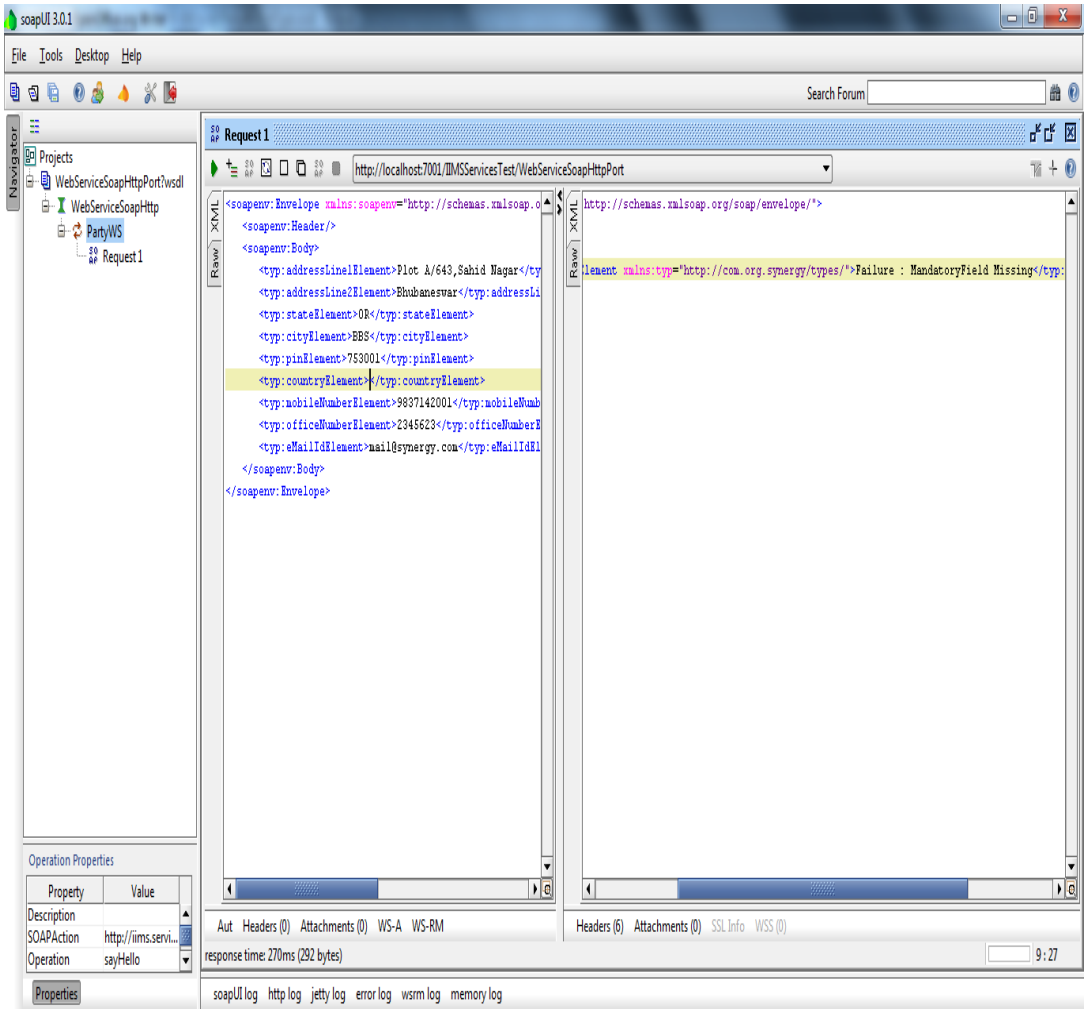

Figure 7: Snapshot of Failure

c) Testing primary "Address details" section of Party Module

Here we narrow down the testing coverage. Instead of testing whole module and associated modules, we only test the modified services of the component. In a nutshell, we have achieved the following.

a) Saving the manpower
b) Saving the time
c) Increasing the scalability
d) Increasing the productivity
e) Test coverage is high within the limited time frame.

Scenario 2:

In this scenario, Let us assume that we have one request from client to remove one functionality from the existing business specification. The requirement is that to remove the miscellaneous section from the presentation layer. Hence, we need to follow the similar steps as we mentioned above with proper documentation and test

case preparation. Once the analysis and design are finished, during the coding only, we need to remove the miscellaneous section (Method4 - Operation name) from the existing web service and then build and generate the new WSDL and deploy in the server. In this case, the testing coverage only would be only to test our services through SOAP UI and regression test case would be to test our miscellaneous section so that we can complete our test scenario in a short span of the project schedule.

Scenario 3:

Here, let us assume that there is a new requirement to add a new functionality in the existing web application. The new requirement is that to send an email to the customer for a particular event like "**party saving"** in the database. Once the party is saved, there would be one email to be sent to that party/customer.

The process is simple to incorporate the changes in the existing system. The steps to be followed are :-

a) Modify the business specification document as per the new requirement
b) Prepare the analysis document
c) Prepare the design document - Low Level
d) Code
e) Prepare unit test case
f) Prepare system test case

In the low level design, we came up with the solution that there is already a web service developed from the client side(Output Management System) which can be reused to meet the business expectation. Here, we have to call their web service from our business layer. In that case, we need to know what inputs are required to call their web service from our business layer. Then, we have to formulate those inputs and call the web service so that it will send the message to the target recipient. The inputs would be

a) ToAddress
b) FromAddress
c) BodyText
d) SubjectLine

The envelope format for the outgoing WSDL is as :

```
<soapenv:Envelopexmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:com="http://com.tcs.bancs.insurance.claim.note">
<soapenv:Header/>
<soapenv:Body>
<com:outgoingMails>
<strFromAddress>XYZCompany.co.in</strFromAddress>
<strToAddress>Customer1@gmail.com</strToAddress>
<strEmailSubject>Test Mail</strEmailSubject>
<strEmailBody>Dear Sir, This is a test.</strEmailBody>
</com:outgoingMails>
</soapenv:Body>
</soapenv:Envelope>
```

So, we can plug-in this code block in our "Party" component and call the service. We need to test only the saving of party/customer in the basic detail section of the PartyWS whether email is successfully sent or not. Besides that, we should not test the other parameters and/or functionalities of the other sections of the party component.

We concluded that addition/deletion/modification of existing component is not creating any risk for doing the regression testing. So, the changes made before and after are always satisfying the business expectations. This will not create any adverse effect after the changes are made, even if it is not being tested from end to end of the system. The reason behind this is that the architecture is fully loosely coupled with the service-oriented design and pattern.

Deriving Regression Test Data

Regression testing measures the relationship between the dependent modules and independent modules and derives the effective changes of the existing application to cover the business functionality test cases. Let us take a comparative study between sections 3 and 5.

Traditional Component Model:

In the section 3, we have mentioned the traditional component based module where all the components are tightly coupled. Now, considering the example of scenario-1 depicted in section 5, we need to add one parameter as "Email ID" in the "Primary Address details" of the party module. In this scenario, we need to find out which modules are affected by these changes and then find out the dependent and independent modules. Based on that we need to plan for the coverage of the testing.

As per the business specification, we have observed the following modules are affected. Here 'M' stands for the module.

1. User Login process – LDAP - M1
2. Application Security module– Authorization/Authentication - M2
3. Party Main Module (M3) and following sub-modules
    - Basic Details - M31
    - Primary Address Details - M32
    - Secondary Address Details - M33
    - Miscellaneous Details - M34
4. Policy Main Modules - M4
    - Policy Quotation Details (During policy quotation Capture) - M41
    - Policy Party Details (During the party association with the policy) - M42
5. Claim Main Modules - M5
    - Claim intimation Details (During claim registered, need to capture the party details) - M51
    - Claim Party Association (Associating party with the claim during claim intimation) - M52

- Claim Party Payment (Giving payment to claim party, system need to know the party details) - M53

6. Reports - Generating the party report - M6
7. User Logging-out -LDAP - M7

Now we derive the dependent and independent modules from above affected modules.

Dependent Modules :M3, M31, M32, M33, M34, M4, M41, M42, M5, M51, M52, M53

Independent Modules :M1, M2, M6, M7

For any changes in the dependent modules of the traditional architecture, regression testing needs to be performed. As depicted in Fig.8, regression testing of independent module starts from T3=30 hours and ends at T8=80 hours. Thus, the total time taken for the purpose is 50 hours and person-days spent for it is 50/8=6.33 considering 8 working hours per day for one person.

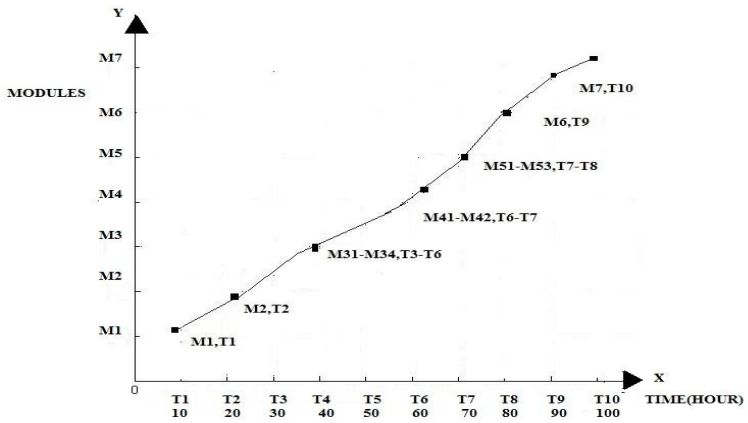

Figure 8: Representation of Modules in Time (Traditional Component Model)

SOA Based Component Model

Now, it can be observed how much time is taken for the same changes through SOA based implementation model. As per above modules and sub-modules, we need to see what are the dependent modules and what are the independent modules through SOA implementations.

Dependent Modules: M3, M32, M4, M5

Here, only the primary section is getting affected and however, we need to check the functionality of the module M4 and M5 as well as to check whether all the changes are impacting or not. (to verify the high level functionalities)

Independent Modules :M1, M2, M31,M33, M34, M41, M42,M51, M52, M53, M6, M7

If we plot the diagram, we can see the overall time taken for the regression testing of only dependent modules [Fig.9].

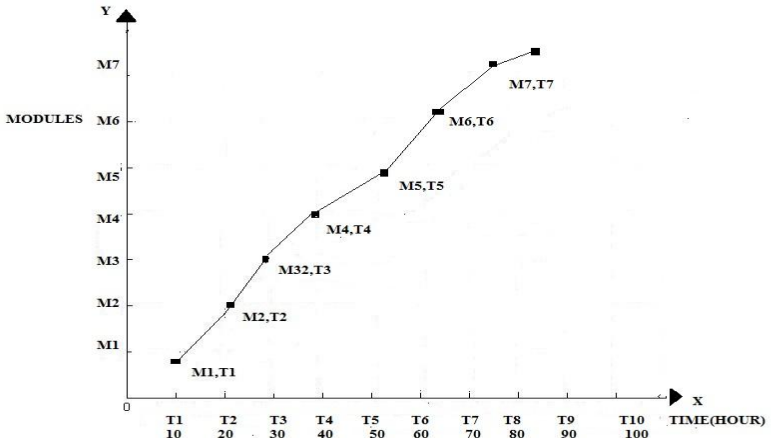

Figure 9: Representation of Modules in Time (SOA Based Component Model)

After SOA implementation of the traditional pattern previously discussed, there are 4 dependent modules considering scenario 1. Hence, we need to perform regression testing of these 4 modules only as listed above. As it can be observed from Fig.9, regression testing of dependent modules starts from T3=30 hours and ends at T5=50 hours and the total time taken is 20 hours. Thus it requires 20/8=2.5 person-days.

Here, it can be observed that the total time taken for traditional component module which covers the testing of the impacted modules is almost 2.5 times more than the SOA integration component modules. The behavioural pattern of the diagram looks like straight line in case of SOA based testing but in the former case it was like a zigzag curve.

Conclusion

In this paper, we have presented an approach of model-based regression test reduction. The approach identifies the difference between the original model and the modified model as a set of elementary model modifications. For each elementary modification, each test case in the regression test suite, interaction patterns are identified based on the business requirement analysis. These patterns are used to reduce the regression test suite. Our initial experience shows that the approach may significantly reduce the size of regression test suites. This model determines the effectiveness of the presented approach and the quality (fault detection capability) of reduced test suites and hence increased productivity.

References

[1] Mac kenzie etal.”OASIS reference model for service oriented architecture 1.0” [online]. http://docs.oasis-open org/soa-rm/v1.o, 2006.

[2] Thomas Erl. “Service Oriented Architecture : Concepts, Technology and Design” Prentice Hall, 2005.

[3] P. Offermann, M. Hoffmann and U. Bub, "Benefits of SOA: Evaluation of an implemented scenario against alternative architectures", Proceedings of EDOCW 2009, pp.352-359, 2009

[4] G.Confora and M.Di Penta “SOA testing and self-checking”, Proceedings of Internal workshop on web services modeling and testing, pp. 3-12, 2006.

[5] Dalal etal “Model Based testing in practice”, Proceedings of the International Conference on s/w engineering, pp.285-294, 1999.

[6] T.A. Khan and R. Heckel, “A Methodology for Model-Based Regression testing of Web Services”, Proceedings of 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques, (2009),

[7] B. Yang, J. Wu, C. Liu and L. Xu, “A Regression Testing Method for Composite Web Service”, Proceedings of the 2010 International Conference Biomedical Engineering and Computer Science, Penang, pp. 1-4, 2010.

[8] A. Tarhini, H. Fouchal and N. Mansour, “Regression Testing Web Service-based Applications”, Proceedings of IEEE/ACM International Conference on Computer Systems and Applications

[9] G. Rothermel, M.J. Harrold. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, Vol.22, Issue 8, pp.529-551,1996

[10] S. Dustdar and S. Haslinger, Testing of service-oriented architectures - a practical approach. 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Lecture Notes in Computer Science, Vol. 3236, pp.97-109, 2004

[11] I. Alsmadi and S. Alda, ‘Test Case Reduction and Selection Optimization in Testing Web Services”, International Journal of Information Engineering and Electronic Business (IJIEEB), Vol.4, No.5, pp.1-8, 2012

[12] A. Tarhini, H. Fouchal, N. Mansour, "Regression Testing Web Services-based Applications", 2006 IEEE International Conference on Computer Systems and Applications, pp. 163-170, 2006