J. Appl. Environ. Biol. Sci., 4(9S)60-69, 2014

© 2014, TextRoad Publication

ISSN: 2090-4274
Journal of Applied Environmental and Biological Sciences
www.textroad.com

On Model Based Regression Testing of Web Services

Tamim Ahmed Khan[1], Umair Maqsood[2], Waqar Ahmed[2], Sadia Ashraf[2], Faisal Aftab[3]

[1]Department of Software Engineering, Bahria University, Islamabad
[2]Department of Software Engineering, [3]Department of Management Sciences, Bahria University, Islamabad

Received: September 12, 2014
Accepted: November 23, 2014

ABSTRACT

Regression testing is helpful in finding out if the quality of an evolved system has not regressed during maintenance. In case of web services, it becomes an issue since access to implementation is restricted to interface level information only. The problems gets further complicated if the testing is manual and the effect of maintenance is minimal. We can establish an idea about the depth of changes through the analysis of exposed interface and provide test oracle by storing test data and returned values in case of successful test cases. This paper proposes an interface analysis approach where we analyze these interfaces written in web service description language (WSDL). We utilize data collected form interface analysis to invoke service operations and develop test oracles. We use these test oracles for regression testing to ensure there is no regression in the operations where the interface has not changed. In case, the interface has changed, we can detect by rerunning interface analysis to find out evolution in the signature.

KEYWORDS: web services, regression testing, test oracles, web services description language (WSDL), reflections.

1 INTRODUCTION

Regression testing in web service is difficult since actual implementation of the web service is only accessible through exposed interfaces and many of the established testing methods cannot be applied owing to this limitation of access to code [3]. Software systems evolve over time or they go obsolete [13] and we need to continuously monitor if the preexisting system quality has not regressed in the evolution process. Testing in case of web services has an additional cost since it involves service invocation [16].

In case of web services exposing their operations by means of WSDL, we can find out details of operations such that both the input and return types can be found out. These may be simple data types or they may be complex data types composed of simple data types. From modeling perspective, complex data types are the objects passed as parameters or returned as return types. Test cases are, on the other hand, composed of inputs and expected return values. Once we know input parameters and their types from WSDL analysis, we can develop test cases and write drivers to execute operations to receive execution output as a response object. If we maintain a record of the operation details, inputs and actual outputs in case of successful executions, we can prepare test oracles for future runs and we we can find out interface level evolutions from interface analysis of consecutive versions. Together these are enough to lead us to know what operations are added, removed or evolved and if there is any regression in operations that have remained the same in two versions.

We propose, in this paper, an approach where we examine exposed interfaces of simple object access protocol (SOAP) based web services written using web service description language (WSDL) to extract operation signatures. Our analysis is capable of extracting simple data types and complex data types that are serializable and involve basic data types in their composition. After extracting operation signatures, we are ready to execute these operations. We use reflections to develop an adapter so that we are able to execute a web service considering extracted operations. We allow the users of our system to select data values so that concrete test cases could be formulated and provide the execution result back to the user for verifying if the returned value was correct. Once the user confirms that the execution response was in accordance with expected response, the input and the actual response are saved for providing a test suite and a test oracle. We call this training data for regression testing step. This way, the user completely tests web service under test (WSUT) and the user can rerun all of the test cases so achieved for regression testing. The activity diagram shown in Figure-1 presents the whole process discussed above.

*** Corresponding Author:** Tamim Ahmad Khan, Department of Software Engineering, Bahria University, Islamabad (tamim@bui.edu.pk)

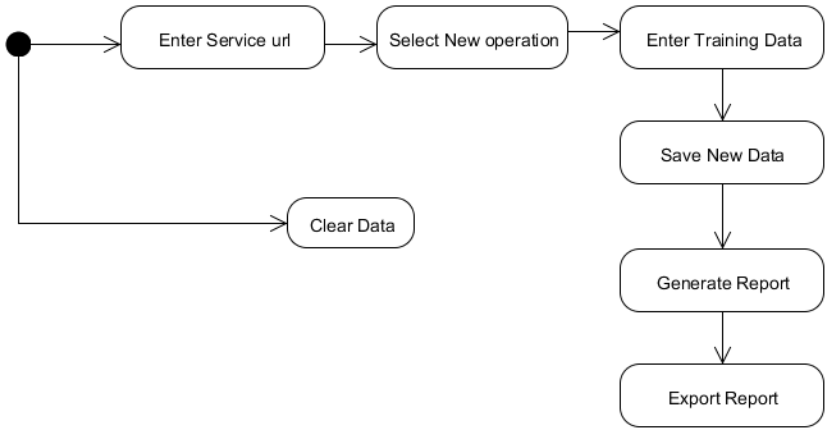

Figure 1: proposed system

The paper is organized as follows. An overview of regression testing techniques is presented in Section 2 and our proposal is introduced in Section 3. We present evaluation of our proposal in Section 4 and discuss related work in Section 5 while the conclusion and outlook is presented in Section 6.

2. REGRESSION TESTING: AN OVERVIEW

Regression testing is retesting of modified parts of system under test (SUT) to see if there is any regression in preexisting system quality [7]. Since regression testing is a process to verify quality of the evolved system, there is a test suite already available to select test cases. Following test case classification given in [14], a test case already available in the test suite can be classified as obsolete, re-testable or reusable. A test case can be regarded as "obsolete" if the evolved system has been updated such that a functionality previously existing is seized to exist or part of code has been merged, replaced or removed from the rest of the code.

Regression testing may be broadly classified into two types - one in which the specifications of the system do not change, termed as corrective regression testing, and other in which the specifications may change, called progression regression testing [14]. The requirement of the former may arise consequently to activities such as corrective maintenance or refactoring whereas the later may be required in response to modifications in existing functionality, adaptation or evolution. Regression test selection techniques for progression regression testing can be classified into the following [6, 11]:

- Retest All Techniques: We select all existing test cases for rerunning and it is useful in case of security critical or high risk systems.
- Ad-hoc/Random Techniques: We rely on human judgment and test cases are selected randomly using expert judgment.
- Safe Techniques: Certain test selection criterion is defined assuming that changes would be covered.
- Minimization Techniques: Modification in the system is calculated e.g., in [8]. Test cases are selected such that they cover all modified parts of SUT.
- Data-Flow/Control-Flow Techniques: Changes in data/control flow information before and after the change is identified to select test cases test modified/added flows.

Apart from the first of the identified techniques cited above, the rest may be termed as selective retesting techniques and may work on the basis of specifications or the code of the system and hence can be divided into two main groups - Code based or White-box or Specifications based or Black-box techniques. A typical selective retesting technique works on the basis of identifying changed part of the program code P' from P, identifying changed part of the specifications S' from S, and selecting subset of existing test cases T' from T such that running of T' on P' may cover complete changed part or a need to write some new cases T" may be felt which may be eventually written and the program P' may be tested for regression using T'UT" [6, 11, 14, 18, 19]. Model-based regression testing techniques consider system specifications or model of a system and try to find out the required test cases for regression testing on the basis of specifications analysis

and are appropriate for web services regression testing. The following section introduces our proposed approach for regression testing of web services.

3. OUR APPROACH

This section is devoted to explaining our proposed approach for arriving at test cases and development of test oracle which becomes a resource for development of test suite for regression testing. We access service interface, written using WSDL as discussed earlier, through its provided URL and parse it for extracting information. We gather required information such as operation name, operation‘s input output parameters are gathered from analysis of exposed interface. This is implemented using reflection class in .Net [17]. After the information is extracted from the WSDL document, we have list of all operations with signatures mentioned in the interface.

Algorithm 1 generation of test oracle

```
Require: URL of the web service to be tested
  select web service from the URL (in case of multiple web services)
  for (i=0; i<num(operations); i++) do
    if Operation is already invoked AND interface is unchanged then
      Automatically rerun operation and report result
    else
      WSDL is parsed, web method is extracted and input/output parameters are extracted
      if test executes successfully then
        update oracle with input data, save actual output (verified by tester) as expected outpu
        for subsequent executions
      else
        discard test case
      end if
    end if
  end for
```

We allow the user to select web method one by one and give the input data and expected output data which formulates a test case. The web service needs to be invoked to run the test case and we need a mechanism for invocation of WSUT to dynamically access operations in web service. We use CodeDOM compiler [5] for client development purposes and automatic running of test cases first time around. The test case are run considering each operation separately and all the results are saved for developed of test oracle.

Theses test cases and the test oracle allow us to do regression testing for subsequent test runs. The system runs all available test cases for regression testing and provides and comparison of results of previously tested functionalities with the results of new results of the same functionalities. The test cases which were previously successful and have failed in the evolved version reported. This is shown in algorithm 1 involved in this process.

Algorithm 2 regression analysis

```
Require: URL of the web service to be tested, test input data and oracle from previous run
  for (i=0; i<num(operations); i++) do
    if test passed, result saved in database then
      if actual output from current run = expected output read from last run data then
        test passed, no regression
      else
        test failed
      end if
    end if
  end for
```

Once interface is analyzed and test oracle is prepared, the tester can analyze subsequent versions of the same web service. This process allows to find out if there is any evaluation seen at the interface level. In order to see its impact on the rest of the operations that have not displayed any evolution at the interface level, test data and oracle from previous runs is used for regression testing. This is shown in algorithm 2.

However, it is important to highlight that the proposed solution is limited in the sense that testing of simple types and first level complex types is possible. That means, we are only able to consider complex

types (objects passed as parameters) that do not further involve complex types and therefore all non-serializable types cannot be tested. Our proposed system is also unable to do complete testing on the collection of complex types as a parameter as well.

4. EVALUATION

We explains our approach on a relatively small application that we developed for test purposes. We develop two versions where an operation is added, one operation is deleted, signature of one operation evolves and one operation remains unchanged. We report operations deleted, added and evolved from signature standpoint and we provide testing output of the forth operation for regression testing result.

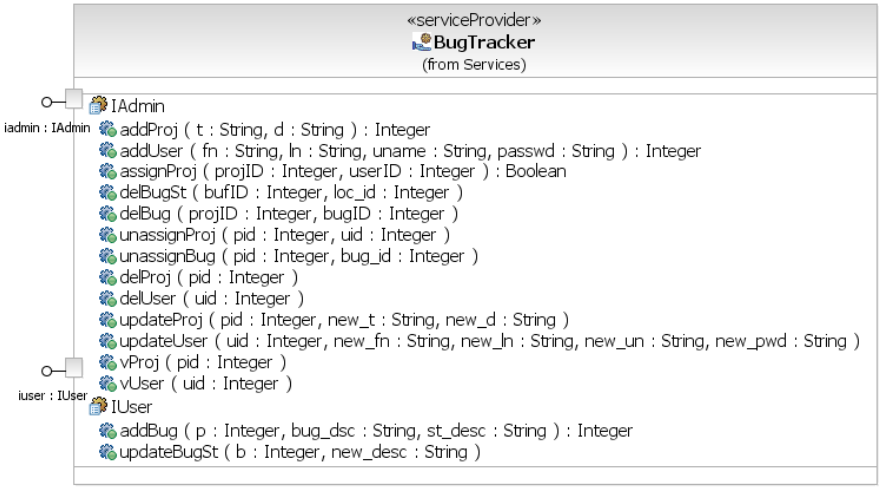

Figure 2: Interface

For detailed analysis, We present a case study which we have developed from an open-source (http://btsys.sourceforge.net/) bug tracking desktop application. We have converted this desktop application into a service-oriented application which, we call bug tracking service (BTS) and, is useful to automate the process of fault reporting. We have also provided an interface for administrative issues e.g., add/update/delete a project and user. The status of the bug i.e., if it is fixed etc can be updated subsequently by the development team so that the test team could test and the end user may get notified. The administrative and user interfaces after evolution as shown in Figure 2. The evolved version of the same system has additional operations to add status if the bug is fixed, updating signatures for project and user to contain more information and the rest of the operations are kept unchanged intentionally. The first run allows us to maintain test oracle where the second run automatically runs tests from previous run to report any regression due to addition and update of operations.

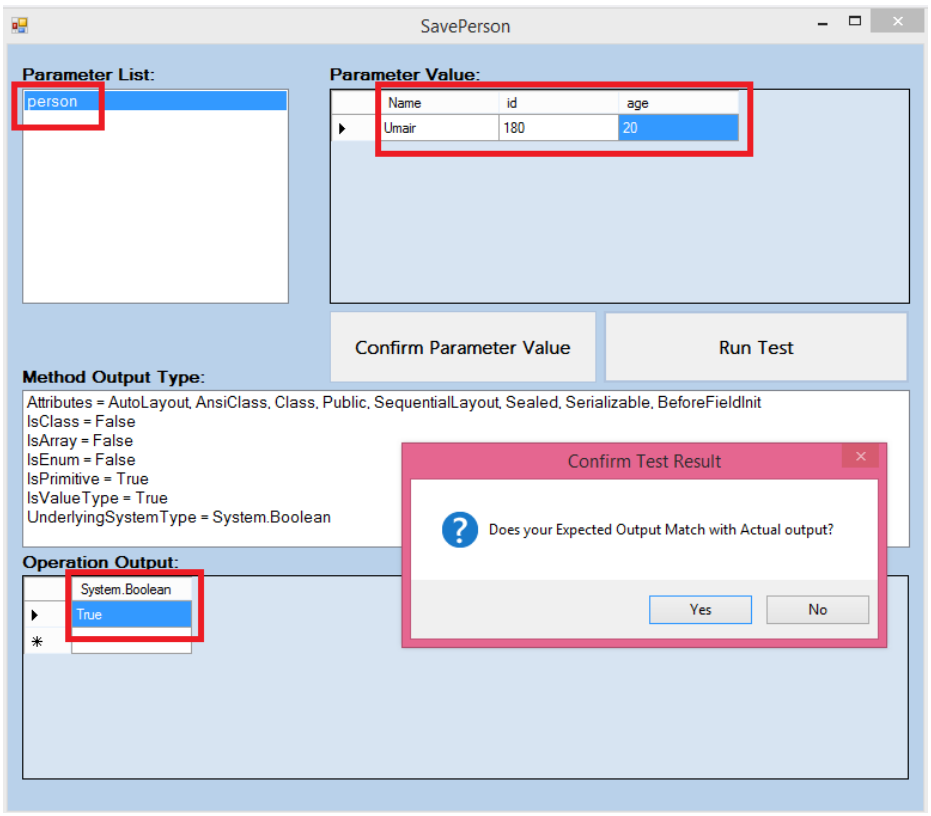

Figure 3: WSDL Analysis Interface

For the first run, we retrieve all operations in the provided interface and we provide tester with list of all input parameters and provide them an interface for entering possible input values. This step could be automated by providing values from a possible range for simple data types. Since we are allowing the user to provide values for complex data types used in the WSUT, we provide an interface for inputs and allow them to investigate return types of operations. The user inputs data required to invoke the operation, the operation is invoked automatically and results are displayed back so that the user can confirm if they were correct. In case the actual result returned is in line with expected output, the test case is recorded and the test oracle is updated. This is shown in Figure 3 where parameter "person" is passed to operation "Save Person" which has name, id and age and the response object contains system response which is of type Boolean and contains "true" as value in it.

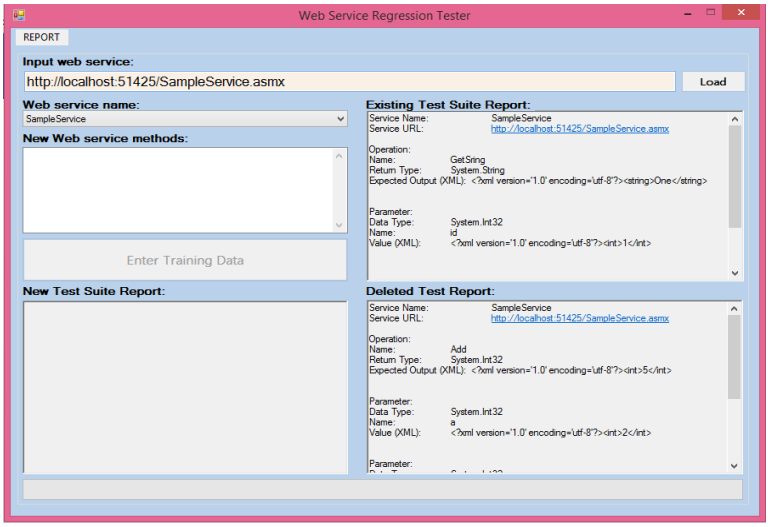

Figure 4: Result Panel

Once a test case execution is successful for a WSUT, an XML file is updated where the information about operation, inputs and actual output is saved. This provides information about operations in a particular release of a WSUT, provision of test oracle and becomes basis for rerunning test cases for regression testing. The subsequent executions use this saved test cases data and test WSUT for regression testing. These executions show which of the test cases were successful, which new operations were added for which new data is required and which of them have seized to exist. This is shown in Figure 4 where the execution reports that no new operations are added, one operation has seized to exist and all previously existing show no regression.

We evaluated our proposed application on the case study discussed earlier in this section. We found out that the approach was useful in the sense that all the test cases and the resulting test oracle was saved in an XML file. However, the approach was taking more time to provide results as the number of operations were growing. A closer analysis of underlying algorithm revealed that the complexity is however not growing exponentially. Another important issue faced was that the system was only able to interpret faults related to system functionality and cases where system should not report a fault, as discussed in [10], were reporting false positives. Therefore, the system was not able to differentiate cases where there were issues such as temporary non-availability of service, timeouts, etc. The approach, however, worked well for web services with average number of the operations and where there were no technical faults such as timeouts, communication failure etc., as discussed in [10]. The output of first run considering application before evolution is shown in Figure 5.

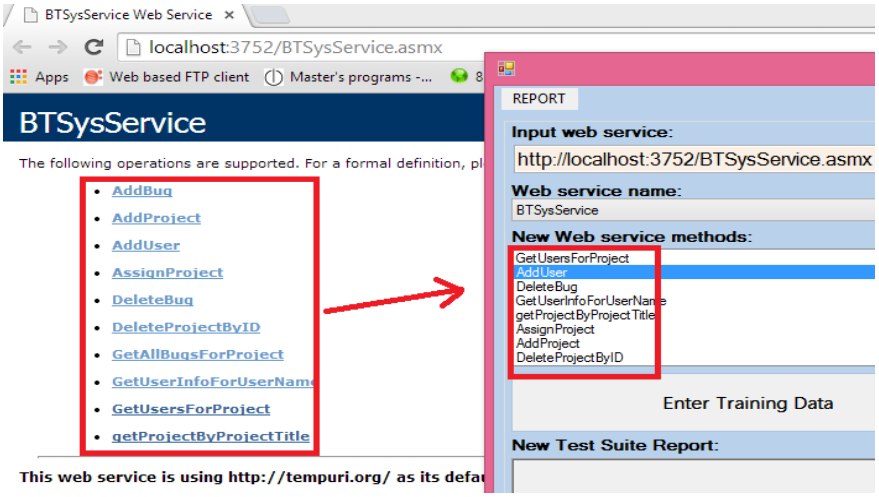

Figure 5: BTS Service Execution

Figure 5 shows how operations visible in service are viewed in our developed tool and Figure 6 shows the fact that all operations were executed for the first time and hence they are visible in the new test cases section of graphical user interface of our application.

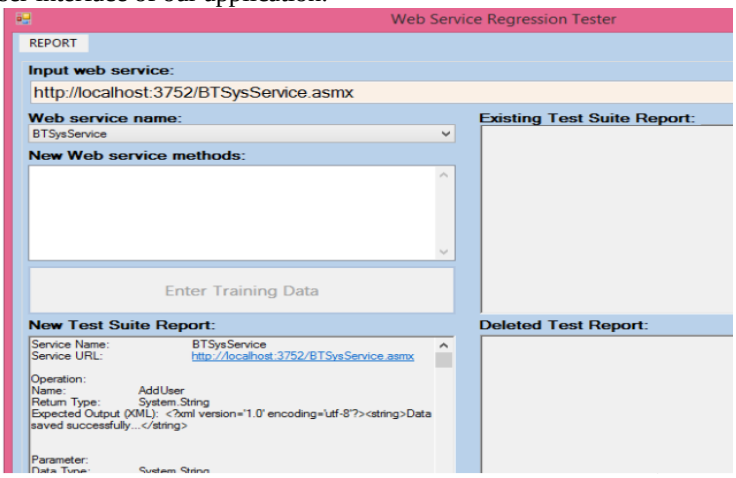

Figure 6: Execution Report

The execution before evolution results in creation of an xml file and a snapshot of resulting xml file is shown in Figure 7 where an operation "GetAll Bugs For Project" is highlighted with a red rectangle showing the input is "projectId" and the output is object set "Bug[]". The operation execution result is saved in the shape of actual output and the testers response is also recorded.

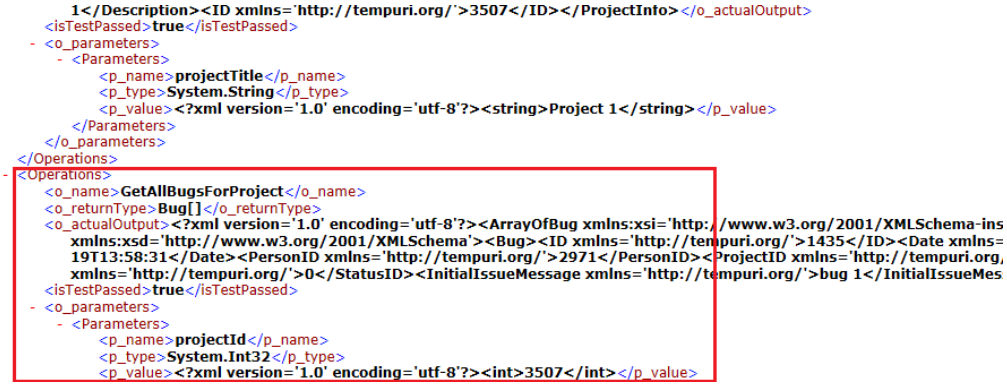

Figure 7: Test Oracle

We evolved BTS service such that new operations were added and some of the operation signature were modified. We tested the application for regression and found out that the developed tool was reporting failure in some operations that had not evolved.

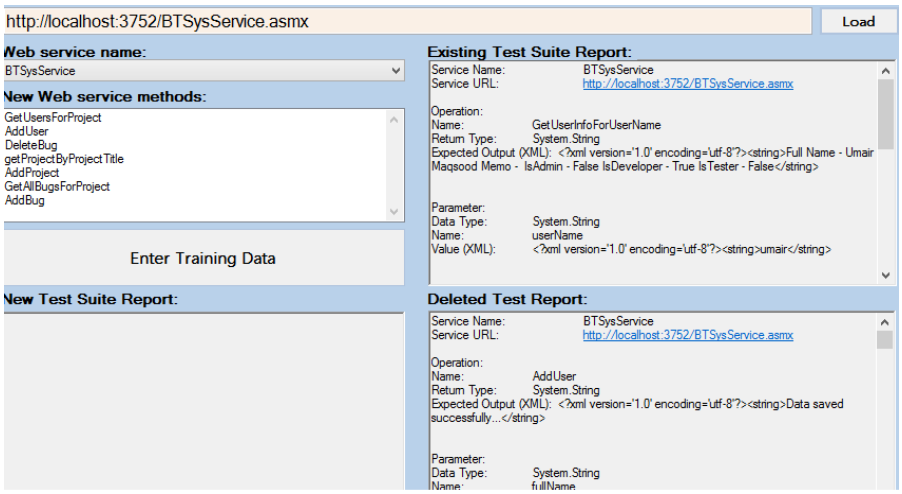

Figure 8: After Evolution Execution Report

A closer look at operations revealed that the system tried to create duplicate database entries as evident in Figure 8 and hence some of the test cases, shown in Figure 9, related to operations such as "Add User" reported failure.

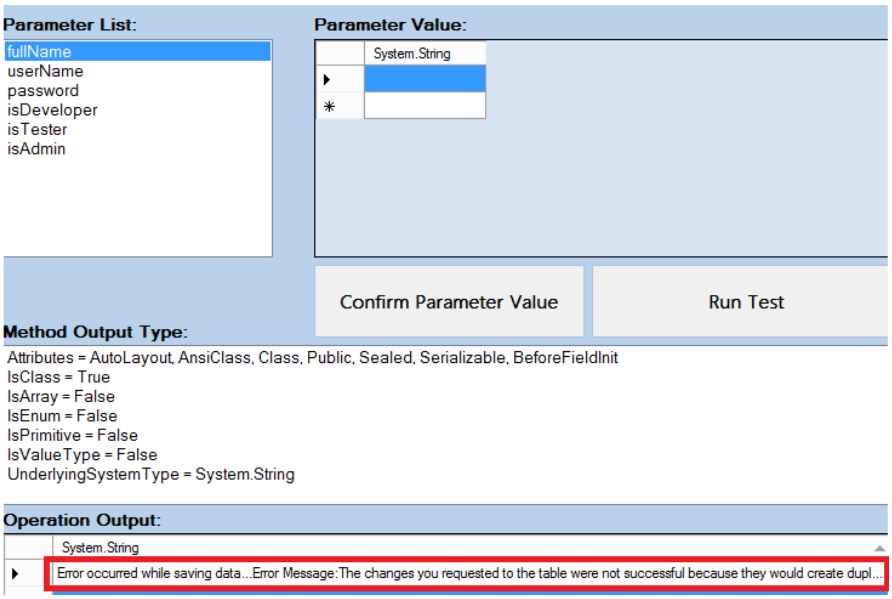

Figure 9: Reason for False Positive

5. RELATED WORK

Regression testing consuming model level information is considered in previously e.g., in [12], where regression test suite is developed considering modifications in the finite state machine (FSM) representation of the system under scrutiny before and after evolution. The authors identify changes in transitions between states and additionally use evolution in data and control dependence while selecting test cases for rerunning. The same concept is extended in [4] where a state dependence graph is annotated with additional transitions to represent control and data dependence and it is tested if a du-pair is affected consequence to a change in recorded transition.

Regression test selection analyzing UML artifacts is proposed in [2] where evolution in artifacts is identified from analysis of associated object constraint language (OCL) expressions. The authors have also developed a tool to present changes in models before and after evolution through analysis of XML representation (XMI) of these artifacts. Web services regression testing considering call graph analysis is proposed in [20] where operation name are used to construct flow graph (CFG) and an analysis is conducted whenever evolution in operations is observed.

Analysis of WSDL for purpose of testing is done, e.g. in [15], where mutation testing approach is introduced and the authors introduce faults and develop mutant versions for testing of web services. A mutant is killed in case output of the mutant version of the operation differs from the original operation. Test case generation from WSDL is also proposed in [1] where WSDL is analyzed not only for generation of test cases but proposing suitable test data. The authors also propose a coverage criteria based on operation coverage, operation flow coverage and message coverage.

Our approach is novel in the sense that we use WSDL analysis to identify evolution in operations presented in two successive versions of a service. We further identify those operations that have been newly added, deleted or evolved considering interface level information. We select those operations that have remained same and we keep track of inputs and actual outputs for last executions of tests on WSUT. This allows us to propose test oracles and a means for automation of test execution as we not only have the input data for test cases but expected output as well. We use reflections to automatically invoke operations while testing.

6. CONCLUSION

Service oriented systems are only accessible through their exposed interface written using WSDL for SOAP-based services. We can develop test cases from the analysis of information available in the interface and maintain invocation history to provide test oracles. These test cases and test oracles are helpful in regression testing of web services since we can find out which of the operations are returning the same

output as before. We can also find out from the analysis of WSDL which of the operations have been added or deleted in the newer versions of a service.

We plan to investigate how can this approach be merged with coverage analysis proposals discussed in [9]. Another line of investigation could be to find out how test oracles could be developed from algebraic specifications, as discussed in [10] to automate development of test oracles which is manual in the current proposal.

REFERENCES

1. Xiaoying Bai, Wenli Dong, Wei-Tek Tsai, and Yinong Chen. Wsdl-based automatic test case generation for web services testing. In Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop, pages 207-212, 20-21 2005.
2. L. C. Briand, Y. Labiche, and S. He. Automating regression test selection based on UML designs. Inf. Software Technol., 51(1):16-30, 2009.
3. Gerardo Canfora and Massimiliano Di Penta. Testing services and service-centric systems: Challenges and opportunities. IT Professional, 8(2):10-17, 2006.
4. Yanping Chen, Robert L. Probert, and Hasan Ural. Model-based regression test suite generation using dependence analysis. In A-MOST '07: Proceedings of the 3rd international workshop on Advances in model-based testing, pages 54-62, New York, NY, USA, 2007. ACM.
5. System.codedom namespace. http://msdn.microsoft.com/en-us/library/system. codedom.aspx, Accessed on April 2, 2014.
6. Todd L. Graves, Mary Jean Harrold, Jung-Min Kim, Adam Porter, and Gregg Rothermel. An empirical study of regression test selection techniques. ACM Trans. Software Engineering Methodologies, 10(2):184-208, 2001.
7. Rajiv Gupta, Mary Jean, Mary Jean Harrold, and Mary Lou Soa. An approach to regression testing using slicing. In Proceedings of the Conference on Software Maintenance, pages 299-308. IEEE Computer Society Press, 1992.
8. Tamim Ahmed Khan and Reiko Heckel. On model-based regression testing of web-services using dependency analysis of visual contracts. In Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011, Held as Part ETAPS 2011, Saarbrucken, Germany, March 26-April 3, 2011. Proceedings, Vol. 6603 of LNCS, pages 341-355. Springer, 2011.
9. Tamim Ahmed Khan, Olga Runge, and Reiko Heckel. Testing against visual contracts: Model-based coverage. In 6th International Conference, ICGT 2012, University of Bremen, Germany 24 - 29 September, 2012, LNCS 7562, pp 279-293.
10. Tamim Ahmed Khan, Olga Runge, and Reiko Heckel. Visual contracts as test oracle in AGG 2.0. In Proceedings of Graph Transformation and Visual Modeling Techniques, GTVMT 12, Electronic Communications of the EASST, Vol. 47, 2012.
11. Jung-Min Kim, Adam Porter, and Gregg Rothermel. An empirical study of regression test application frequency: Research articles. Software Testing Verification Reliability, 15(4):257-279, 2005.
12. B. Korel, L.H. Tahat, and B. Vaysburg. Model based regression test reduction using dependence analysis. In Software Maintenance, 2002. Proceedings. International Conference on, pages 214-223, 2002.
13. M. M. Lehman. Laws of software evolution revisited. In EWSPT '96: Proceedings of the 5th European Workshop on Software Process Technology, pages 108{124, London, UK, 1996. Springer-Verlag.
14. H.K.N. Leung and L. White. Insights into regression testing [software testing]. In Software Maintenance, 1989., Proceedings., Conference on, pages 60-69, Oct 1989.

15. Nashat Mansour, Reda Siblini, and Abbas Tarhini. Wsdl mutation for testing web services. I. J. Comput. Appl., 12(4):210-216, 2005.

16. Massimiliano Penta, Marcello Bruno, Gianpiero Esposito, Valentina Mazza, and Gerardo Canfora. Web services regression testing. pages 205-234. 2007.

17. System.reection namespace. http://msdn.microsoft.com/en-us/library/system.reflection.aspx, Accessed on April 2, 2014.

18. Gregg Rothermel and Mary Jean Harrold. Analyzing regression test selection techniques. IEEE Transactions on Software Engineering, 22, 1996.

19. Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. ACM Trans. Software Engineering Methodologies, 6(2):173-210, 1997.

20. Michael Ruth, Sehun Oh, Adam Loup, Brian Horton, Olin Gallet, Marcel Mata, and Shengru Tu. Towards automatic regression test selection for web services. In COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference, pages 729-736, Washington, DC, USA, 2007. IEEE Computer Society.