Enhancing Robustness of WCF Web Services

Vijaysinh G. Chavan[1] Subhash V. Pingale[2] Priyanka S. Mangale[3]
[1,2,3]Assistant Professor
[1,2,3]Department of Computer Science Engineering
[1,2,3]SKN Sinhgad College of Engineering, Korti, Pandharpur, Solapur University, Solapur, India

Abstract—Web service is a collection of open protocols and standards used for exchanging data between applications or systems. There are large numbers of web services deployed with robustness problems. Unexpected behaviour of the system in the presence of invalid input is called as robustness problem. Robustness testing is used to detect robustness problems in web services and then reduces those issues by applying inputs verification based on well-defined parameter domains. This paper proposes approach to improve the robustness of web services.

Key words: Web Service, Interoperability, Reliability, Robustness and Testing

I. INTRODUCTION

Web services are a popular way of implementing a Service-Oriented Architecture. There are three major roles within the web service architecture: Service Provider, Service Requester and Service Registry. Service provider is the provider of the web service. The service provider implements the service and makes it available on the internet. Service requester is any consumer of the web service. The requester utilizes an existing web service by opening a network connection and sending an XML request. Service registry is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services. Here service requester is distributed application builder.

Web service provides a simple interface between provider and consumer. Provider provides set of services that are used by consumer. Provider must provide robust web service to consumer.

These interactions involve publish, find and bind operations.

A. Publish

In this operation, service provider published service description. It needs to be accessible so that the service requester can find it, where it is published can vary depending upon the requirements of the application.

B. Find

The service requester finds service description directly or requests the service registry for such kind of service required.

C. Bind

A service needs to be invoked. In the bind operation the service requester invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

The web services background is split into three areas — communication protocols, service descriptions, and repair discovery. Web service is an implementation of SOA that is based on set of XML based technologies such as SOAP and WSDL. Simple Object Access Protocol (SOAP) which plays the role of messaging protocol for exchanging data between service provider and service requester. It is stateless and one way message exchange standard. Web Service Description Language (WSDL) is used to describe web service as collection of communication endpoints that can exchange particular type of message. It provides a formal, computer readable description of web service. It describe interface of service and provide users. Universal Description, Discovery and Integration (UDDI) are registry of web service. It provides a mechanism for clients to discover web service. It is used to find service providers through a centralized registry [8].

Testing web services for robustness is an effective way of disclosing software bugs. Robustness is defined as the degree to which system operates correctly in the presence of exceptional input. Robustness testing is testing methodology to detect the vulnerabilities of component under unexpected input. It is efficient and effective technique which characterizes the behavior of the system.

Previous work on web service robustness testing shows that services are being deployed with robustness problem, which require robustness testing and improvement methodology.

II. LITERATURE REVIEW

Nuno Laranjeiro, Marco Vieira and Henrique Madeira have mentioned a Technique for Deploying Robust Web Services. This technique gets all the information about the web service thoroughly by getting all the list of parameters, operations, data types and parameter domain. This will create workload and execute the workload by choosing one or more generating strategies of workload. Also create a service workload. After executing the created workload, it will be verified by the service. After fixing the robustness of web service will verify the behavior of service to identify potential deviation [1].

Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer gave overview of Fault Injection Techniques and Tools that focused on Hardware fault injection technique and Software fault injection technique. In Hardware fault injection uses additional hardware to introduce faults into the target system's hardware. In Software fault injection, inject faults at compile-time; the program instruction must be modified before the program image is loaded and executed. Rather than injecting faults into the hardware of the target system. During runtime, a mechanism is needed to trigger fault injection [2].

M. Vieira, N. Laranjeiro, and H. Madeira have presented idea about Assessing Robustness of Web-Services Infrastructures. They presented.

1) Analysis of the WSDL of the services under testing in order to identify the relevant call parameters.
2) Robustness tests generation based on the information gathered in the previous step.
3) Execution of the web services without fault injection in order to collect baseline performance metrics and to understand what behavior is expected from each service.
4) Execution of the web services in the presence of invalid call parameters (robustness tests). The goal is to trigger faulty behaviors, and in that way expose robustness problems.
5) As a final step the results obtained in 3) and 4) are used to characterize the performance and robustness of the web services infrastructure.

By using this we can reveal robustness problem. But not applicable to external web service response [3].

N. Laranjeiro, M. Vieira, and H. Madeira gave Improving Web Services Robustness. This mentioned following steps to improve robustness of web service.

– Gather required information and prepare the web service.
– Workload generation and execution
– Execute a set of robustness tests
– Fix disclosed robustness problem and verify the service behavior.

In this benerator tool is used to read xml file. But this does not implement all generation strategy. EDEL code is used to represent domain information, but this EDEL code is complex [4].

K.M. Senthil Kumar, A.S. Das, and S. Padmanabhuni provided an overview of WS-I Basic Profile: A Practitioner's View they have used to examine some of the key interoperability issues that are encountered in SOAP 1.1 and WSDL 1.1 implementations by practitioners. Also analyze how BP addresses these issues [5].

III. PROPOSED SYSTEM

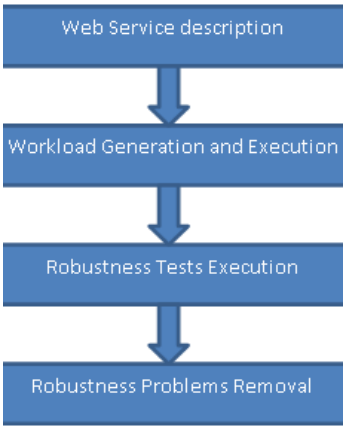


Fig. 1: Proposed System

A. Web Service Description

Firstly, collect all information about operation, parameter & associates data types by processing WSDL file or SVC file of WCF Web Service. Information about structure and data types of all input and output is usually found in those files.

Secondly, collect information about all valid domains of input and output object by searching XS file. Also this file includes information about valid values of each parameter.

B. Workload Generation and Execution

In this step, generation of workload is needed to exercise each operation which is provided by web service.

At starting point using XSD file we are able to generate workload means set of valid web service calls. (Stage 1)

In next step, we need to create programming language level object that represent structure found in XSD file to use generated values. (Stage 2)

We form our final workload by creating list of object by de-serializing produced XML into corresponding generated object. (Stage 3)

By using reflection, we load classes by name and builds list of objects that are integrated into one unit test case per each operation. (Stage 4)

The stages 1, 2, 3, and 4 can be performed with the help of WCF Test Client which is discussed in next section. Also code coverage will be 100%. Because if we do not make a call to any one of the service operation then the message log will not be considered for creating a WCF test as given the next section.(Stage 5)

At last, we execute the final workload to understand the behavior of service in presence of valid input values. This result is used to identify possible deviation from expected output domain.

C. Robustness Test Execution

Robustness test is executed to identify robustness issues. If we require redefining parameter domain then return to earlier step.

D. Robustness Problem Removal

In this step, all invalid incoming requests are aborted and exception is thrown.

To identify remaining problem, re-execute robustness test. Also we re-execute workload to verify service answer to identify potential deviation.

IV. RESULT ANALYSIS

A. Test Data

The 7 WCF services and corresponding web service clients were developed to provide an input to this project. They are as given below:

1) Library Management Service
2) Eschool Service
3) Ebanking Service
4) Chatting Service
5) Real Estate Service
6) Email Sending Service
7) Simple Calculator Service

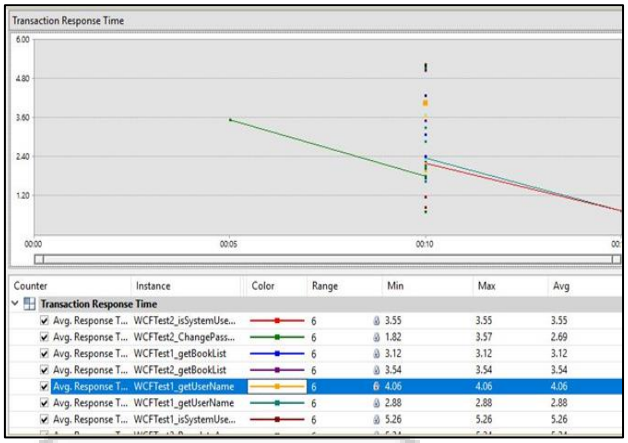
Also 3 open source services were considered as an extra input for this project:

1) Message Queuing Service
2) Simple Sale Product Service
3) File Repository Service

B. Testing

Testing was done with help of WCF Test Client [11] included with Microsoft Visual Studio 2010. Initially the path of service contract file (svc file) was given as an input to test client. Client configuration was modified so that it will listen for service message logs with the help of svc configuration editor.

The client configuration file generated after editing the svc configuration editor after enabling message logging and activity tracing for a library service invocations is given below:



Graph 1: Average Response Time for Service operations of Library Service under the load test

```
<? Xml version="1.0" encoding="utf-8"?>
<Configuration>
<system. Diagnostics>
<Sources>
<source name="System.ServiceModel.MessageLogging"
Switch Value="Warning, ActivityTracing">
<Listeners>
<add type="System.Diagnostics.DefaultTraceListener"
name="Default">
<filter type="" />
</add>
<add name="ServiceModelMessageLoggingListener">
<filter type="" />
</add>
</listeners>
</source>
<source name="System.ServiceModel"
switch Value="Warning,ActivityTracing"
propagateActivity="true">
<listeners>
<add type="System.Diagnostics.DefaultTraceListener"
name="Default">
<filter type="" />
</add>
<add name="ServiceModelTraceListener">
<filter type="" />
</add>
</listeners>
</source>
</sources>
<sharedListeners>
<add
initializeData="D:\svclogs\Client.dll_messages.svclog"
type="System.Diagnostics.XmlWriterTraceListener,
System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
name="ServiceModelMessageLoggingListener"
traceOutputOptions="LogicalOperationStack, DateTime,
Timestamp, ProcessId, ThreadId, Callstack">
<filter type="" />
</add>
<add initializeData="D:\svclogs\Client.dll_tracelog.svclog"
type="System.Diagnostics.XmlWriterTraceListener,
System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089"
Name="ServiceModelTraceListener"
traceOutputOptions="LogicalOperationStack, DateTime,
Timestamp, ProcessId, ThreadId, Callstack">
<filter type="" />
</add>
</sharedListeners>
</system. Diagnostics>
<system.serviceModel>
<diagnostics>
<messageLogging logEntireMessage="true"
logMalformedMessages="true"
logMessagesAtServiceLevel="true"
logMessagesAtTransportLevel="true" />
<endToEndTracing propagateActivity="true"
activityTracing="true"
messageFlowTracing="true" />
</diagnostics>
<bindings>
<basicHttpBinding>
<binding name="BasicHttpBinding_IGlobals" />
</basicHttpBinding>
</bindings>
<client>
<endpoint address="http://localhost:4159/Globals.svc"
binding="basicHttpBinding"
bindingConfiguration="BasicHttpBinding_IGlobals"
contract="IGlobals"
name="BasicHttpBinding_IGlobals" />
</client>
</system.serviceModel>
</configuration>
```

As given in the above service configuration service message log will be recorded in the file with a path given as “D:\svclogs\Client.dll_messages.svclog” and activity tracing log will be recorded in the file with a path given as “D:\svclogs\Client.dll_tracelog.svclog”

Each service operation was invoked separately with allowed values and with null values or unintended values to carry out the robustness tests. According to those invocations the service message log was generated. It was found that if any one of the service operation is not invoked then generated log will be rejected in the next step and we have create log again by invoking all service operations. This guarantees the complete code coverage.

Support for WCF test was added using WCF Load Test utility [12]. The Visual Studio C# test project was created. To add WCF test to the same project generated service log and DLL file of service contract is used. Next to that Load test was added with a scenario of more than 100

clients and scenario was run to get the robustness issues and problems.

A Library Service was developed with a Windows Application (.exe) client. So in the process of adding the WCF test to the test project, service message logs were recorded by running an exe client of Library Service. Added WCF test was used for creating a Load Test with a scenario of more than 100 users using step load of 10 users. Average response time graph for each service operation of Library Service is given on previous page.

For the service operations those are not supported by WCF Test Client (Message Queuing, File Repository, Chatting), manual testing is done to detect the robustness issues.

After testing robustness an issue were detected and according to that if service code is causing the problem then source code was immediately modified.

Finally the issues such as null checks, poor login management etc. will be handled by generating a wrapper service. Unit test log for major service operations of Library Service is given below.

1) Unit Test 1

Test Name: Database connectivity
Test case input: Call to any database operation without making the initialization of Library Service members i.e. without making call to populateAppData ()
Module Associated: All service operations except populateAppData () e.g. issueBook ()
Purpose of the test: To make sure that the data is accessible for required operation.
Result: FaultException1 was unhandled by user:
"ConnectionString property has not been initialized."
Action taken: The source code of web service was modified and repeated the test again to get the correct output.

2) Unit Test 2

Test Name: Adding books
Input: Details of book to be added (Title, Author, Publisher, etc.)
Module Associated: Adding books to the library database.
Purpose of the test: To make sure that the book details are successfully added to the database.
Result: Book was added to the library but generated an error that "Key is already present in dictionary."
Action taken: The source code of web service was modified and repeated the test again to get the correct output.

3) Unit Test 3

Test Name: Issue a previously issued book
Module Associated: issueBook () service operation
Purpose of the test: To ensure that the same book is not issued multiple times to a single user.
Result: Same book is issued to multiple times to a single user.
Action taken: The service wrapper will be developed to tackle this issue and the test will be repeated to get the correct output.

4) Unit Test 4

Test Name: Register a new user
Input: Details of new user for registration
Module Associated: registerUser () service operation
Purpose of the test: To ensure that the user is successfully registered with given input.
Result: User registered successfully.

5) Unit Test 5

Test Name: Registering the same user again
Input: Details of previously registered user for registration
Module Associated: registerUser () service operation
Purpose of the test: To ensure that the user is registered only once to avoid duplicates.
Result: Duplicates were not allowed and exception was generated:" Violation of UNIQUE KEY constraint 'IX_Lib_Users' cannot insert duplicate key in object 'dbo.Lib_Users'. The statement has been terminated."
Action Taken: Exception will be handled and proper user friendly message will be generate.

6) Unit Test 6

Test Name: Login to the library system
Input: User login credentials
Module Associated: ValidateId () service operation
Purpose of the test: To ensure only user with correct login credentials will get access to the system.
Result: Test successfully completed, but customized client can directly bypass the login.
Action Taken: Login management will be improved using the wrapper service.

7) Unit Test 7

Test Name: Changing the user password
Input: username, new password
Module Associated: ChangePassword () service operation
Purpose of the test: To make sure that changing a password task works well.
Result: Changing password is completed successfully, but a customized web service client can change the password without having the old password.
Action taken: ChangePassword () service operation will be improved in wrapper service.

8) Unit Test 8

Test Name: Removing a book from the library
Input: Details of book to be removed (BookTitle, Quantity, AuthorName, and CategoryName)
Module Associated: RemoveBook () service operation
Purpose of the test: To make sure that book can be removed from library with given details.
Result: Book removed successfully.

9) Unit Test 9

Test Name: Searching a book from the library
Input: Details of book to be searched (BookTitle, CategoryName)
Module Associated: SearchBook () service operation
Result: Search book operation returns dataset containing a table with book details matching with BookTitle and/or Category.

10) Unit Test 10

Test Name: Initializing the book issued details
Input: Call to PopulateReturnBooks () service operation.
Module Associated: PopulateReturnBooks () service operation
Purpose of the test: To make sure that ReturnBooks data is initialized, so that ReturnBook () service operation will go through.
Result: The operation successfully initializes the ReturnBooks data.

11) Unit Test 11

Test Name: Returning an issued book.

Input: Details of book to be returned (BookId, IssueId)
Module Associated: ReturnBook () service operation
Purpose of the test: To make sure that the book is returned successfully.
Result: Book is successfully returned.

12) Unit Test 12

Test Name: Check whether a book is already issued to user.
Input: Details of book to be issued.
Module Associated: ValidateBook () service operation
Purpose of the test: To make sure that whether a book is already issued to user.
Result: Gives correct output.

As given in above test log, the tests are performed for detecting the issues and they will be solved in the next step by creating the wrapper for a WCF Service.

V. CONCLUSIONS

This paper proposes approach, which is used for improving the robustness of WCF web services. It includes complete flow of testing WCF services for robustness issues. Also transaction response time for each service operation is included by using Visual Studio Load Test scenario. Test results generated by executing the scenario will definitely help in analyzing the behavior of each service operation separately. Test log shows that there is a scope for improvement even if the service can work correctly with legal inputs. Also poor login management is observed in most of the services, which is big risk for the service operation invocations. Because anyone can invoke the operations and make the changes to the database without having user login credentials. Also most of service operations have unexpected behavior, in case of null or unintended inputs to the service operation. All these problems will be solved by creating the wrapper for the services having robustness issues. Additional argument checking will be provided using a wrapper for such services. So that all unintended or illegal inputs will not be allowed to reach the actual service operation and the risk of such invocations will have no effect on the server hosted data.

ACKNOWLEDGMENT

I am profoundly grateful to Prof. V. G. Chavan. For his expert guidance and continuous encouragement throughout to see that this paper rights its target since its commencement to its completion.

I like to express deepest appreciation towards Prof N. M. Sawant, (Coordinator of ME CSE) SKN SINHGAD COLLEGE OF ENGINEERING whose valuable guidance supported us in completing this paper.

At last I must express our sincere heartfelt gratitude to all the staff members of Computer Engineering Department who helped me directly or indirectly during this course of work.

REFERENCES

[1] Nuno Laranjeiro, Marco Vieira and Henrique Madeira : "A Technique for Deploying Robust Web Services", IEEE Transactions On Services Computing, Vol. 7, No. 1, January-march 2014.
[2] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer : "Fault Injection Techniques and Tools", Computer, vol. 30, no. 4, pp. 75-82, Apr. 1997.
[3] M. Vieira, N. Laranjeiro, and H. Madeira: "Assessing Robustness of Web-Services Infrastructures", Proc. IEEE/IFIP 37th Ann. Int'l Conf. Dependable Systems and Networks, pp. 131-136, 2007.
[4] N. Laranjeiro, M. Vieira, and H. Madeira: "Improving Web Services Robustness", Proc. IEEE Int'l Conf. Web Services, 2009. R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, "High-speed digital-to-RF converter," U.S. Patent 5 668 842, Sept. 16, 1997.
[5] K. M. Senthil Kumar, A.S. Das, and S. Padmanabhuni: "WS-I Basic Profile: A Practitioner's View", Proc. IEEE Int'l Conf. Web Services, pp. 17-24, 2004.
[6] V. Santiago, A.S.M.D. Amaral, N. L. Vijaykumar, M.D.F. Mattiello- Francisco, E. Martins, and O.C. Lopes: "A Practical Approach for Automated Test Case Generation Using State charts", Proc. 30th Ann. Int'l Computer Software and Applications Conf., pp. 183-188, 2006.
[7] C. Fetzer and Z. Xiao, HEALERS: "A Toolkit for Enhancing the Robustness and Security of Existing Applications", Proc. IEEE/IFIP Int'l Conf. Dependable Systems and Networks, pp. 317-322, June 2003.
[8] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana: "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI", IEEE Internet Computing, vol. 6, no. 2, pp. 86-93, Mar. /Apr. 2002.
[9] M.Doliner,"Cobertura",http://cobertura.sourceforge.net/ , 2010.
[10] Atlassian, "Clover-Code Coverage Analysis", http://www.atlassian.com/software/clover/, 2010.
[11] MSDN,"WCFTestClient",https://docs.microsoft.com/en us/dotnet/framework/wcf/wcf-test-client-wcftestclient-exe, 2017.
[12] Visual Studio ALM Rangers, "WCF Load Test",https://wcfloadtest.codeplex.com/releases/view/47044, 2010.

