An Adaptive Framework for Web Services Testing Automation Using JMeter

Conference Paper · November 2014

DOI: 10.1109/SOCA.2014.36

1 author:

Srinivas Shenoy
Malaysian Institute of Microelectronic Systems
4 PUBLICATIONS **3** CITATIONS

An adaptive framework for web services testing automation using JMeter

Srinivas Shenoy, Nur Asyikin Abu Bakar, Rajashekara Swamy
Information Communication Technology Division
MIMOS Berhad.
Kuala Lumpur, Malaysia
tsrinivas.shenoy@mimos.my, asyikin.bakar@mimos.my, rajashekara.swamy@mimos.my

Abstract **—Web services testing have become important in Service oriented architecture (SOA). Several organizations have developed generic components that have been used across multiple customer project implementations. They require extensive human interventions for evaluating and testing web services which may include new development and enhancement of components. We propose a new test automation framework for web services testing by aggregating common and project specific features of generic components. The proposed system does not require human intervention in contrast to existing systems where human intervention is necessary. Besides, the proposed system reduces test data dependency, which in turn speeds up execution process. We test the framework on different applications to show that the proposed framework is effective in terms of time and success rate.**

Keywords —Web service testing, Automation framework, WSDL, SOAP messages, JMeter.

I. INTRODUCTION

A test automation framework is an integrated system that sets the rules of automation of a specific product which integrates several components such as function libraries, test data sources, object details and various reusable modules [15]. These components help building a suitable automation framework which enable testing the business process as per the requirements. In addition, test automation is used to control the execution of tests and the comparison of actual outcomes with predicted outcomes. Therefore, test automation framework is useful in several applications. For example, one such application is web service testing, where it is designed to support interoperable machine to machine interaction over a network. According to literature, we can classify existing systems web services into two types broadly as follows [2].

“Big” Web Services: An Extensible Markup Language (XML) language defines message architecture and message formats. Big web services use XML messages that follow the Simple Object Access Protocol (SOAP) standard. Such systems often contain a machine readable description of the operations offered by the service. It is written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically. SOAP based design need to include a formal contract need be established to describe the interface that the web service offers. WSDL can be used to describe the details of the contract, which may include messages, operations, bindings, and the location of the web service.

Representational State Transfer (RESTful) Web Services: The functionality for RESTful web services is well suited for basic, ad hoc integration scenarios. RESTful web services, often better integrated with HTTP than SOAP based services are, do not require XML messages or WSDL service API definitions. It is noted that in all web service applications, the designed testing automation framework does not work efficiently as it requires human intervention and test data dependency [14]. Therefore, there is a great demand for designing an automation framework which does not require human intervention and reduces test data dependency to speed up the process which in turn improves efficiency of system and productivity of the system.

There are several web service testing systems developed in the past [13], namely, SOA , which provides [3] application functionality as services to other applications. This is known as service orientation. It is independent of any vendor, product or technology. Service oriented enterprise architecture incorporates service, composition, and inventory architectures, plus any enterprise wide technological resources accessed by these architectures. This can be further supplemented by including enterprise-wide standards that apply to the aforementioned architecture types. SOA's main goal is to deliver agility to business. Services can be orchestrated [4] for implementing long-running business processes. In the web services platform architecture, web service orchestrations are described using the Business Process Execution Language (BPEL), which are in turn exposed as web services. A typical visual notation used to describe these processes is the Business Process Modeling Notation (BPMN). Web services separate [5] out the service contract from the service interface. This feature is one of the many characteristic required for an SOA-based architecture. It is clearly a great enabler for SOA. Web services are hardware, platform, and technology neutral [12]. The producers and/or consumers can be swapped without notifying the other party, yet the information can flow seamlessly. An Enterprise service bus (ESB) can play a vital role to provide this separation.

Binding Web Services: A web service's contract is specified by its WSDL and it gives the endpoint details to access the service. When we bind the web service again to an ESB, the result will be a different endpoint, which we can advertise to the consumer. When we do so, it is very critical that we don't lose any information from the original web service contract [17].

Functional and non-functional web service testing [1] is done with the help of WSDL parsing and regression testing is performed by identifying the changes made thereafter. Web service regression testing needs can be categorized in three different ways, namely, changes in WSDL, changes in code, and selective re-testing of web service operations.

Software testing execution is laborious and time consuming if it is done manually [6]. A manual approach might not always be effective in finding certain classes of defects. Test automation offers a possibility to perform these types of testing effectively. Once automated tests have been developed, they can be run quickly and repeatedly. Regressions testing of software products have long maintenance duration [11]. Even minor patches over the lifetime of the application can cause existing features to break which were working at an earlier point in time.

From the above review of existing frameworks, it is observed that none of the existing systems work perfectly for the different applications as they have inherent limitations and weakness. Thus, in this work, we propose a new design of adaptive framework for testing automation applications to web-services. The proposed adaptive framework makes use of JMeter [10] which is a 100% pure Java desktop application [7] found to be very useful and convenient in support of functional testing in different way to design adaptive framework for testing automation in this work. Although JMeter is known more as a performance testing tool, functional testing elements can be integrated within the Test Plan. In this way, out system uses JMeter in different way to achieve objectives.

The rest of the paper is organized as follows: section two highlights Proposed Framework, section three highlights Experimental Results, Section four emphasize on Conclusion and Future Work.

II. Proposed Framework

The proposed framework is divided into four stages, namely, thread groups selection, driver script development, adaptive script development, and act on system under test. First, we select the thread group based on each web services to minimize the number of thread groups to make it generalized. Second, driver script can be SOAP or Java Message Service(JMS) or REST sampler which accepts maximum number of test cases with generic sampler based on available test data. Third step consists of three types of functions: generic, project specific and results. Generic functions read test data from Comma separated values (CSV) or any other files. It also declares generic variables for system properties. Project Specific functions are used when situation demands (For example, size of payload, controllers etc.). Results functions are used to write the outcome in HyperText Markup Language(HTML), XML, CSV formats. Fourth, Act on system under test receives the payload format from sampler and then it send response payload back to sampler after processing.

The block diagram of the generic framework is shown in Figure 1 where all four stages are depicted in logical order. It is developed and tested with multiple platforms. In this work, we test this generic framework on one of the platform as a case study. Framework of the case study is show in Figure 2 where Information Dissemination System (IDS) web services automation architecture is shown. The functionality of each component in IDS web service architecture is given below. (1) Jmeter tester executes the IDS scripts using Jmeter/Ant plugin/Maven plugin [8]. (2) The system control moves to first thread group belongs to first web service. i.e. Send Email. The thread group will be executed in a loop for a specific period (3) Thread group invokes Driver script. Driver script (sampler for test cases) shown in Figure 2 will execute test cases in a sequential pattern. (4) The sampler or driver script uses either generic or project specific functions for further processing. (5) The sampler or driver script read data from CSV files (Generic - URL and project specific) as input. This will generate the standard payload request format. (6) Scripts send payload request format generated from previous step to system under test (SUT), in our case IDS platform. SUT process and returns back the SOAP response to Jmeter sampler. Sampler will process the response using the assertions (Response and Size Assertions). Then JMeter generates Result using Results functions (listeners) like Simple data writer, view results tree etc.. (7) The generated results are storing in the Result.csv file, .jtl and .xml formats. Step 2 to 7 will be executed for all the thread groups. IDS has 16 services which translates to 16 thread groups. In addition, it has 5 exceptional thread groups as well. In total it executes for 21 thread groups. (8) If the user has Ant/Maven for script execution, the result will be stored in HTML format. Ant/Maven plugin uses .jtl file as input and process to generate the HTML report.

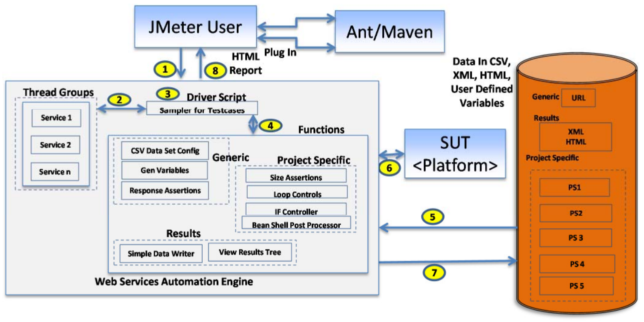


Figure 1: Functional test flow environment - Adaptive

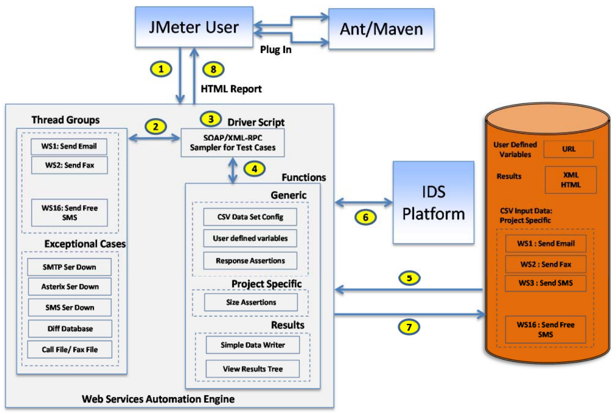


Figure 2: Functional test flow environment – IDS Case Study

A. Thread Group Selection

As shown in Figure 2, each web service has one thread group and one sampler. The main function of thread group selection is to cover both positive and negative test cases [16]. "Bundle of test cases in one web service that have same XML payload will be shared via one sampler." This helps in selecting minimum number of thread groups, samplers and test data. Overall, the thread group, sampler and test data selection help us to obtain optimized scripts with high maintainability. For example, WS1: Send Email which sends email and it is one of the email web services. The send email web service thread group with one samplers parameterized (TC Name, URL, value of SOAP parameters email Type, email, Content) may run multiple test cases. It can run total 11 test cases which involves positive and negative. One thread group and one sampler help in optimized script. Parameterization helps in maintainability as all data is read from CSV file. There are total six email web services. Similarly for WS2: Send Fax is one of the Fax related web services. The Send Fax web service thread group with one sampler parameterized may run multiple test cases. It can run multiple types of test cases both positive and negative. However exceptional test cases are handled differently sometimes. For example, when Simple Mail Transfer Protocol (SMTP) server is down, thread group considers all six emails as related exceptional test cases.

B. Sampler – Several Request Type

Section 2A provides background sampler usage in the framework. In the Figure 2 we demonstrated with SOAP sampler. However, three categories of samplers are available namely SOAP, JMS and REST. SOAP uses JMeter sampler - SOAP/XML-RPC Request. All individual platforms used this sampler. We add the SOAP body with automation customization here. For example, value of a field will be parameterized to read the data from .CSV file specific cell. Similarly, JMS uses JMeter sampler - JMS Point-to-Point or JMS Subscriber or JMS Publisher. This is used with platform and ESB combination in the implementation. Similarly, REST uses JMeter sampler and config element combination of HTTP Request Defaults, HTTP Header Manager, HTTP Request and Java Request based on the web service functionality. This has been used with BPM workflow invocation for framework implementation.

C. Functions Description

Section 2A and 2B briefed on sampler in detail. Samplers use many functions in generating the request payload format. Functions help in payload or request processing with right data fetching from input, writing data to the output file or processing the output. Functions are categorized as three categories, namely Generic, Project Specific and Results. Generic Functions have been used in all the platforms, mostly for all test cases. For example, (1) CSV Data Set Config helps in declaring variable, fetching the value for each iteration and assigning value to variable. CSV file contains all input data like expected results together with variable name. Jmeter is able to invoke the data that needed in every test case as input. One CSV file caters one web service. (2) User Defined Variables are generally used in all the test cases in the platform For example, URL. (3) Response Assertions compare the expected results (reads from file using variable) with actual result received after IDS Platform. The result will be PASS if response and expected result variable are matched. The result will be FAIL if response and expected result variables do not match. Project Specific Functions are used only for few test cases in the IDS platform. For instance, Size Assertions are used only for three web services like BatchAsynchronousEmail, BatchAsynchronousSMS and BatchAsynchronousFax. This is used for checking the expected response payload size matches with actual payload received from IDS platform. The results functions help in writing the results in various formats for easy analysis of results. For example, (1) Simple Data Writer writes all response and request data and it generates assertion result in XML file format. This is used for analyzing the failed test cases during result analysis. (2) View Result Tree shows full results of all sample responses. CSV file will be generated if produce path in filename field. This will be used extensively during script development or enhancement for debugging purpose. This function implementation will help in portability, Reusability, maintenance and faster script development.

D. Table Description

In Section 2C briefed Generic, Project Specific and Results functions. These functions use Tables (CSV files) extensively. This is very important section because we ensure that all test data are separated from the test case (sampler – Section 2A and 2B). Test Data is stored in the Table like CSV file. Any change in data in subsequent test releases requires changes only in test data. The changes to scripts are eliminated. For Example (1) WS1: Send Email CSV has all variables used in Send Email web service are declared as the table header. Positive and negative input test data for each test case is available as one record for one test case. For Example, EmailType, Email, Content, ExpectedResult etc. (2) WS2: Send Fax CSV has all variables used in Send Fax web service are declared here as the table header. Positive and negative input test data for each test case is available as one record for one test case. For Example, FaxType, PhoneNo, Content, ExpectedResult. The same is applicable to the remaining 14 CSV files.

III. EXPERIMENTAL RESULTS

In this section, we demonstrate the proposed test automation framework to show its effectiveness and usefulness in terms of qualitative results and analysis.

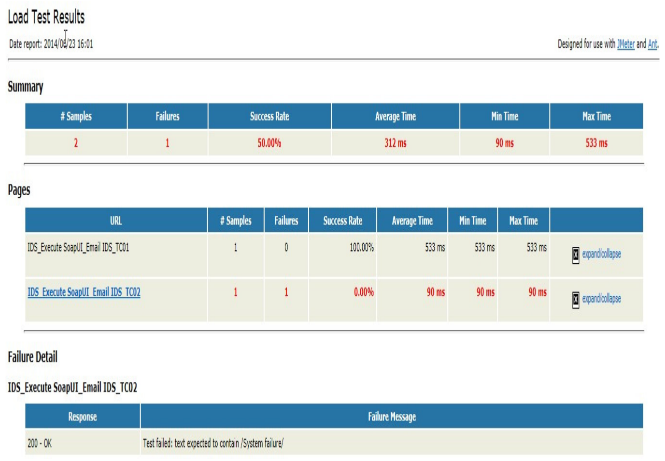
A. Software and Licenses Requirements

The Automation framework requires the minimum software tools like apache-jmeter-2.11 with plugin ant-jmeter-1.1.1.jar, apache-ant-1.9.3 and jdk1.6.0_21(jre6) or above.

B. HTML Report using Ant Plugin

In the framework, JMeter and Ant integration is conducted using a plugin. This helps us in executing the JMeter script using Ant. It generates HTML format report. Table 2 represents the report with an example. For easy understanding we take only two test cases execution simultaneously. The report shows 2 samples (test cases) are executed, one failure and one success results. During result analysis, we can click the failure test case which shows hyperlink. This provides the Failure details like SOAP Response and Failure message. This could be used for further analyzed. This report is a test repository asset to be stored in configuration management tool for process standard in each future releases.

Table 2: HTML Report using Ant execution



Load Test Results

Date report: 2014/06/23 16:01

Designed for use with JMeter and Ant.

Summary

# Samples	Failures	Success Rate	Average Time	Min Time	Max Time
2	1	50.00%	312 ms	90 ms	533 ms

Pages

URL	# Samples	Failures	Success Rate	Average Time	Min Time	Max Time	
IDS_Execute SoapUI_Email IDS_TC01	1	0	100.00%	533 ms	533 ms	533 ms	expand/collapse
IDS_Execute SoapUI_Email IDS_TC02	1	1	0.00%	90 ms	90 ms	90 ms	expand/collapse

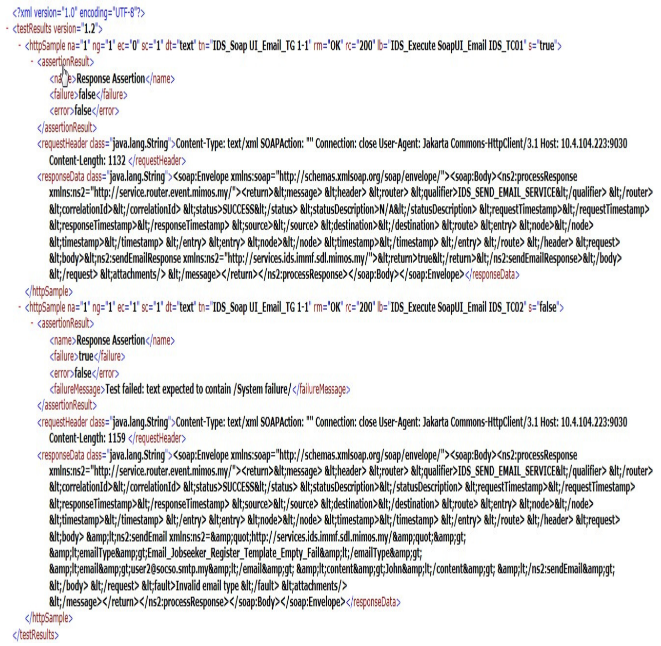
Failure Detail

IDS_Execute SoapUI_Email IDS_TC02

Response	Failure Message
200 - OK	Test failed: text expected to contain /System failure/

C. Experimental Results in XML from Simple Data Writer

Listener Simple Data writer is described in the proposed framework. This is used to generate the XML report during execution. One such example is shown in Figure 4. This contains test case id, request and response payload, execution result, assertion results and failure details. As suggested in section 3B further analysis is conducted using this report. During report analysis and defect reporting phase, failure result in subsection 3B, various details of Figure 4 and various details are useful for identifying defect or application enhancement or script improvement. If it is a defect, both sections provide details that are required for defect reporting in a defect tracking tool. Tester raises the defect using this.



```
  <?xml version="1.0" encoding="UTF-8"?>
- <testResults version="1.2">
  - <httpSample na="1" ng="1" ec="0" sc="1" dt="text" tn="IDS_Soap UI_Email_TG 1-1" rm="OK" rc="200" lb="IDS_Execute SoapUI_Email IDS_TC01" s="true">
    - <assertionResult>
        <name>Response Assertion</name>
        <failure>false</failure>
        <error>false</error>
      </assertionResult>
      <requestHeader class="java.lang.String">Content-Type: text/xml SOAPAction: "" Connection: close User-Agent: Jakarta Commons-HttpClient/3.1 Host: 10.4.104.223:9030
        Content-Length: 1132 </requestHeader>
      <responseData class="java.lang.String"><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:processResponse
        xmlns:ns2="http://service.router.event.mimos.my/"><return>&lt;message> &lt;header> &lt;router> &lt;qualifier>IDS_SEND_EMAIL_SERVICE&lt;/qualifier> &lt;/router>
        &lt;correlationId>&lt;/correlationId> &lt;status>SUCCESS&lt;/status> &lt;statusDescription>N/A&lt;/statusDescription> &lt;requestTimestamp>&lt;/requestTimestamp>
        &lt;responseTimestamp>&lt;/responseTimestamp> &lt;source>&lt;/source> &lt;destination>&lt;/destination> &lt;route> &lt;entry> &lt;node>&lt;/node>
        &lt;timestamp>&lt;/timestamp> &lt;/entry> &lt;entry> &lt;node>&lt;/node> &lt;timestamp>&lt;/timestamp> &lt;/entry> &lt;/route> &lt;/header> &lt;request>
        &lt;body>&lt;ns2:sendEmailResponse xmlns:ns2="http://services.ids.immf.sdl.mimos.my/">&lt;return>true&lt;/return>&lt;/ns2:sendEmailResponse>&lt;/body>
        &lt;/request> &lt;attachments/> &lt;/message></return></ns2:processResponse></soap:Body></soap:Envelope></responseData>
    </httpSample>
  - <httpSample na="1" ng="1" ec="1" sc="1" dt="text" tn="IDS_Soap UI_Email_TG 1-1" rm="OK" rc="200" lb="IDS_Execute SoapUI_Email IDS_TC02" s="false">
    - <assertionResult>
        <name>Response Assertion</name>
        <failure>true</failure>
        <error>false</error>
        <failureMessage>Test failed: text expected to contain /System failure/</failureMessage>
      </assertionResult>
      <requestHeader class="java.lang.String">Content-Type: text/xml SOAPAction: "" Connection: close User-Agent: Jakarta Commons-HttpClient/3.1 Host: 10.4.104.223:9030
        Content-Length: 1159 </requestHeader>
      <responseData class="java.lang.String"><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><ns2:processResponse
        xmlns:ns2="http://service.router.event.mimos.my/"><return>&lt;message> &lt;header> &lt;router> &lt;qualifier>IDS_SEND_EMAIL_SERVICE&lt;/qualifier> &lt;/router>
        &lt;correlationId>&lt;/correlationId> &lt;status>SUCCESS&lt;/status> &lt;statusDescription>&lt;/statusDescription> &lt;requestTimestamp>&lt;/requestTimestamp>
        &lt;responseTimestamp>&lt;/responseTimestamp> &lt;source>&lt;/source> &lt;destination>&lt;/destination> &lt;route> &lt;entry> &lt;node>&lt;/node>
        &lt;timestamp>&lt;/timestamp> &lt;/entry> &lt;entry> &lt;node>&lt;/node> &lt;timestamp>&lt;/timestamp> &lt;/entry> &lt;/route> &lt;/header> &lt;request>
        &lt;body> &amp;lt;ns2:sendEmail xmlns:ns2=&amp;quot;http://services.ids.immf.sdl.mimos.my/&amp;quot;&amp;gt;
        &amp;lt;emailType&amp;gt;Email_Jobseeker_Register_Template_Empty_Fail&amp;lt;/emailType&amp;gt;
        &amp;lt;email&amp;gt;user2@socso.smtp.my&amp;lt;/email&amp;gt; &amp;lt;content&amp;gt;John&amp;lt;/content&amp;gt; &amp;lt;/ns2:sendEmail&amp;gt;
        &lt;/body> &lt;/request> &lt;fault>Invalid email type &lt;/fault> &lt;attachments/>
        &lt;/message></return></ns2:processResponse></soap:Body></soap:Envelope></responseData>
    </httpSample>
</testResults>
```

Figure 4: Experimental Results in XML from Simple Data Writer

D. Test Metrics

Based on section 3C we have executed batch script execution for all the platform implemented and consolidated the report. The proposed automation framework is implemented for IDS, Decision Support System (DSS), Authorization and Role Management Component (ARMC), Business process management (BPM) and ESB pass through with each platform combination. The framework is extensively tested with a significant amount of test cases. We execute the batch of test cases to collaborate the results. The results are reported in the Table 3. For example, IDS platform in which total 183 test cases are available. We automate total 152 test cases which achieves 83 % automation. During execution, we confirm all 152 test cases. Table 3 shows that 31 test cases automation consume a significant amount of time. Therefore, we make them block and execute manually in future. In total, out of 671 test cases, we automate 534 test cases which give 80% automation. In case of execution, total 515 test cases are executed successfully, 19 failure and 80 were blocked. Based on the release requirement, selected 80 test cases are executed manually for regression testing.

Table 3: Consolidated Test Metrics for all platforms

Platform Name	No of Test cases	No of test cases automated	% of automation	Pass	Fail	Block
IDS	183	152	83%	152	0	31
DSS	126	73	58%	70	3	0
ARMC	60	56	93%	51	5	0
BPM+ESB	72	72	100%	67	5	0
IDS+ ESB	117	69	59%	69	0	48
DSS+ ESB	29	29	100%	29	0	0
ARMC+ ESB	84	83	99%	77	6	1
Total	671	534	80	515	19	80

E. Test Effectiveness

The proposed automation framework is implemented for many platforms and ESB integration with platform using pass through scenario. The manual and automated test execution time is calculated as shown in Table 4. For Example: IDS platform manual execution time for 152 test cases is 7 days per person. However for automated execution for 152 test cases is 0.8 days per person. It is an effort savings of 6.2 days per release. Total 534 test cases manual execution time is 38 days per person. However for automated execution for it is 4.4 days per person. It is an effort savings of 33.6 days per release. From this we can conclude there is significant amount of time is saved by automated execution of scripts.

Table 4: Comparison of Test Effectiveness

Platform Name	Test Cases Automated	~ Automated execution time (in days)	Manual execution time (in days)
IDS	152	0.8	7
DSS	73	0.8	5
ARMC	56	0.8	6
BPM + ESB	72	0.5	5
IDS + ESB	69	0.5	6
DSS + ESB	29	0.5	3
ARMC+ESB	83	0.5	6
Total	534	4.4	38

IV. Conclusion and Future Work

In this work, we have proposed new test automation framework for web services using JMeter open source test tool. The new testing automation framework includes four stages, namely, thread groups selection, driver script development, adaptive script development, and act on system under test. We evaluate our proposed framework effectiveness in terms of time and success rate. Experimental results show that proposed system works well under different situations. The scope of the proposed approach is limited to sanity or regression testing.

In future, we are planning to extend this to automated test case generation. It will be compared with existing systems to show its superiority. In addition, we are planning to extend this to performance engineering as well.

Acknowledgment

The authors sincerely thank Dr. Shivakumara P, Senior Lecturer, University of Malaya for mentoring in research methodology and paper review.

The authors sincerely thank the MIMOS Berhad, Malaysia, for financially supporting the Software Engineering Lab, for doing extensive research.

References

1. http://en.wikipedia.org/wiki/Web_service
2. http://docs.oracle.com/javaee/6/tutorial/doc/giqsx.html
3. http://en.wikipedia.org/wiki/Service-oriented_architecture
4. https://bpt.hpi.uni-potsdam.de/pub/Public/GeroDecker/wesoa2007-bpmn2bpel4chor.pdf
5. http://www.packtpub.com/article/binding-web-services-in-esb-web-services-gateway
6. http://en.wikipedia.org/wiki/Test_automation
7. http://www.packtpub.com/article/functional-testing-with-jmeter
8. http://javaoraclesoa.blogspot.com/2013/09/sending-html-reports-of-jmeter.html
9. http://softwaretestingexpert.com/2013/10/08/how-to-generate-html-report-in-jmeter/
10. http://jmeter.apache.org
11. M. I. Ladan, "Web Services Testing Approaches: A Survey and a Classification", Springer-Verlag Berlin Heidelberg, pp. 70–79, 2010.
12. N. E. Ioini, "Web Services Open Test Suites", IEEE World Congress on Services, 2011.
13. C. Atkinson, F. Barth, D. Brenner and M. Schumacher, "Testing Web-Services Using Test Sheets", Fifth International Conference on Software Engineering Advances, 2010.
14. S. H. Kuk and H. S. Kim, "Robustness Testing Framework for Web Services Composition", IEEE Asia-Pacific Services Computing Conference (IEEE APSCC), 2009.
15. F. M. Besson, P. Moura, F. Kon and D. Milojicic, "Rehearsal: a framework for automated testing of web service choreographies", Brazilian Conference on Software: Theory and Practice, 2012.
16. A. Tarhini, H. Fouchal and N. Mansour, "A Simple Approach for Testing Web Service Based Applications", 5th International Workshop (IICS), 2005.
17. Srinivasa Tekkatte Shenoy, Akshyansu Mohapatra, Rajashekara Swamy, "A Multi-Dimensional Testing Approach for an ESB", International Journal of Software Engineering and Technology Vol 1, No 2 (2014)