Testing Web Service’s Compositions Following TASSA Methodology

Denitsa Manova, Sylvia Ilieva, Dessislava Petrova-Antonova

Abstract: Service compositions are increasingly used for the fast development of loosely-coupled dynamic distributed applications. Providing their quality is a challenging task due to various problems at infrastructure, service and orchestration levels as well as necessity of performance of different types of testing in unknown context and environment. This paper proposes a methodology for testing web service compositions, described with Business Process Execution Language for Web Services (WS-BPEL). Its feasibility is proved through a case study, showing how the methodology’s steps can be performed through automated tools in order to achieve functional, performance and robustness testing.

Key words: Testing methodology, Testing Web Service Compositions, SOA, WS-BPEL.

INTRODUCTION

In recent years Service-Oriented Architecture (SOA) is becoming increasingly popular and preferred paradigm in development of loosely coupled distributed systems using widely accepted standards. Reducing costs and increasing revenues is fundamental issue to successful enterprises that require flexible and fast connections with their trading partners. Such business interactions are possible via standards-based, interoperable and reusable software components such as web services that are the preferred way for implementation of SOA.

The SOA implementations are promising, but they bring many challenges related to quality assurance and testing. Although various approaches and tools for SOA testing have been proposed, most of them provide partial solutions covering single testing activities such as test path analysis, test case generation, web service emulation, fault injection and so on. However, in order to facilitate the testing process, it is important to integrate all testing activities in a common testing methodology. Therefore, the current research efforts focus on the development of testing methodologies on top of the existing software platforms and tools for SOA testing. Following this research direction, we have developed our TASSA (Testing As Service Software Architecture) methodology for testing web service orchestrations, described with Business Process Execution Language for Web Services (WS-BPEL) [4]. This paper validates the proposed methodology by its application for functional and robustness testing of concrete WS-BPEL process.

The next three sections outlines the problems addressed by the proposed methodology, the methodology itself as well as the existing methodologies in the testing domain. After, a case study showing the usage of the methodology is presented. The last section concludes the paper and gives directions for future work.

MOTIVATION

Testing web service compositions is more complex than testing conventional software systems due to various problems that can be found at a service, orchestration and infrastructure levels. Its implementation requires orchestration of web services that are built and deployed on heterogeneous infrastructures and platforms. Sometimes these services wrap the functionality of legacy systems and components that are outside the organizations’ boundaries and therefore are very hard to be tested. Furthermore, they could be unavailable for a given period of time or in the worst case could be undeployed by their provider. This in turn leads the necessity of emulation of the missing web services.
Additional efforts are needed for generation of appropriate message data, which will replace the actual one expected by the underlying composition. Since the source code of the composed web services is not available, and the test cases should be defined according to public interfaces usually described with Web Service Description Language (WSDL), it is difficult to achieve a high level of test coverage as well as to determine the causes of failures. In addition the web services themselves can participate in several compositions that should be taken into account when establishing the testing environment and executing test cases. The problem becomes more complicated, when dynamic web service binding is implemented. In such case, it is hard to identify which web service is executed at a given moment. The distributed and heterogeneous nature of web service compositions causes difficulties in providing reliability, robustness, accuracy, etc. Therefore in order to achieve a high level of quality, a new testing approaches and methodologies should be developed and applied both on a service as well as on a composition level.

The proposed methodology addresses the challenges described above by focusing on solving the following major problems:
- Impossibility for instrumentation of web services controlled by external providers;
- Lack of automation for testing unexpected behaviour of the web service compositions in case of problems (message delays, communication interruption, etc.);
- Missing or temporally unavailable web services that participate in a composition;
- Lack of test case generation support in order to achieve a required test coverage;
- Difficulties in identifying the causes of failures.

**RELATED WORK**

This section presents briefly a review of the existing methodologies for SOA testing. They are compared according to supported testing level, testing type, application phase and the formalism used for description of the main test artefact. The results from the comparison are presented in Table 1. The table cells marked with dash sign indicate missing information about the corresponding comparative feature.

<table>
<thead>
<tr>
<th>Methodology</th>
<th>Testing level</th>
<th>Type of testing</th>
<th>Application phase</th>
<th>Formalism</th>
</tr>
</thead>
<tbody>
<tr>
<td>[8]</td>
<td>Orchestration, System</td>
<td>-</td>
<td>Design</td>
<td>C-LTS (Label Transition System)</td>
</tr>
<tr>
<td>[9]</td>
<td>Service, Orchestration, System</td>
<td>Functional, Performance Security, Operability</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TASSA methodology</td>
<td>Service, Orchestration</td>
<td>Functional, Performance Robustness, Scalability</td>
<td>Development, Testing</td>
<td>WS-BPEL</td>
</tr>
</tbody>
</table>

*Table 1. Comparison of the existing methodologies for SOA testing*
In [8] a set of recommendations that provide a successful approach for SOA testing are presented. The SOA testing is proposed to be performed on different phases that considered separately. For each phase a specific testing technique is defined. For example, the service-component test phase requires functional, performance, interoperability, backward compatibility, compliance and security testing. The methodology in [1] presents risks, challenges and benefits of SOA related to testing methods. It proposes six types of testing that can be applied to every SOA development phase. The considered types of testing are functional testing, performance testing, interoperability testing, compatibility testing of client application, SOA standards compliance testing, security testing. In [3] PLASTIC platform for testing and verification of functional and non-functional properties of service-based applications is described. The platform includes a set of tools and techniques that can be used alone or in combination depending on specific application scenario. The proposed approaches are applicable to all stages of the service lifetime. Most of the presented methodologies [7, 1, 3, 8, 9] share similar ideas. In contrast in [6] a platform independent approach for specification of tests is presented. It includes three phases: design, generation and execution. The platform relies on a model-based approach, which reduces the time and effort to organize the testing process. The design of the tests using U2TP platform is extended with additional stereotype message. The execution of tests can be performed by JUnit, BPELUnit, jfcUnit and others.

The TASSA methodology, presented in the last row of the Table 1, supports testing at service and orchestration level. It can be used for functional as well as non-functional testing. Currently, the non-functional testing is achieved through performance, robustness and scalability testing. The main test artefact is a business process, described with WS-BPEL. Although the proposed methodology covers similar testing levels, testing types and application levels, it has advantages in three main directions. First, the methodology is extensible with respect to types of testing. Second, it is supported by an automated framework that integrates a set of software tools performing particular methodology’s steps. Last, but not least the methodology relies on WS-BPEL that is widely accepted standard for web service orchestrations, proposed by OASIS consortium.

**METHODOLOGY FOR TESTING WEB SERVICE COMPOSITIONS**

This section describes the TASSA methodology for testing web service compositions. It consists of the following steps:

1) Selection of BPEL process to be tested.
2) Identification of its partner web services.
3) Selection of test strategy that determines how test scenarios should be generated in order to satisfy given test coverage.
4) Selection of test scenario.
5) Identification of process variables and constants, which affect the execution of selected scenario.
6) Generation of abstract test cases for the scenario.
7) Definition of concrete test cases from the abstract ones, containing input values that satisfy the execution of the process according to the selected scenario and are consistent with the data types defined in its XSD schema.
8) Determination of the expected results for the test cases.
9) Execution of the test cases and collection of test results.

Some of the partner services of the process under test could be unavailable for a given period of time or under development. Also their execution in testing environment could require additional payments to their providers. In such situations the above methodology proposes simulation of the missing or unavailable web service behaviour through execution of the following additional steps after step 2:

A1) Identification and formal description of the invoke activities that correspond to the
simulated partner web services.

A2) Generation of appropriate data, which will replace the actual one expected by the process from the simulated partner web services.

A3) Creation of transformed version of the original BPEL process where the all unavailable or missing partner web services are simulated.

If reliability or robustness testing should be performed the following additional steps should be executed after step 2:

B1) Formal description of the faults that will be injected in the process in order to simulate unexpected behaviour.

B3) Creation of transformed version of the original BPEL process with injected faults described on the previous step.

The proposed methodology is implemented by several software tools that perform:

- Generation and execution of test cases for a given business process;
- Isolation of business process from one or more services;
- Fault injection causing unexpected behaviour of the business process;
- Data dependency analysis.

A detailed description of the methodology can be found in [4]. The tools are integrated into a common automated platform, presented in [5].

![Fig. 1 TASSA methodology](image_path)

**CASE STUDY**

In order to validate the feasibility of the proposed methodology, we have defined a case study using a simple but realistic business process called “Check Payment”. It is a part of frequently used business process for online shopping. “Check Payment” calls four external web services involved in the following interaction. First the user sends his personal data and information for the payment. Then a partner service checks if his credit card number is valid. In case of invalid credit card number an error message is sent to the process, otherwise the process continues its normal execution. After, another web service verifies the user e-mail, and if it is incorrect an exception is thrown, otherwise the process goes on. The third web service does currency conversion and the last web service gets the city and state according to the postal code provided with the request. Finally, the process returns information for the payment.

First the functional testing of the “Check Payment” process will be performed. Thus, the steps from 1 to 9 will be applied. Next, in order to validate the steps from A1 to A3 the
behaviour of the “Addresslookup” web service will be simulated. Finally, the robustness of the “Check Payment” will be tested in case of message delay from the “ValidatorDemo” web service. Thus the steps from B1 to B2 of the methodology will be covered.

Following the methodology described so far and using the TASSA framework several test scenarios that verify the behaviour of the business process are defined. Due to the limited space, we will present one of them, including the following steps:

1) Selection of “Check Payment” as the business process to be tested.
2) Identification of partner web services of the BPEL process. The BPEL process invokes four web services, described above.
3) Selection of test strategy. The possible execution paths in “Check Payment” process are four. In order to achieve full path coverage, all of them should be tested.
4) Selection of test scenario that covers branch containing the assign activity “CityAssign” from the tree representation of the business process.
5) Identification of process variables and constants from which the execution of the selected branch depends on. This is performed using Data Dependency Analysis Tool that produces the following result:

6) Generation of abstract case through Test Case Generation and Execution Tool for the operation “OrderPartnerOperation”. Due to the limited space in the paper the abstract test case is not shown, but its structure can be seen below from the concrete one.

7) Specification of test data compliant with the XSD schema of the process for the abstract test case. This step can be performed manually or using Value Generation Tool integrated in TASSA framework, called WS-TAXI [2]. The following input data is specified for the test case:

8) Definition of test assertions using Test Case Generation Tool as follows:

- HTTP Status Code assertion – the expected code is 200, that is a success;
- Check that the SOAP response or the BPEL variable contains regular expression – the expected one is <ord1:city>.*</ord1:city>.
- Assertion for the XPath value – check that the BPEL variable ValidateEMailOut contains XPath expression message/part/ValidateEMailResponse/ValidateEMailResult with value true;
- Assertion for the XPath value – check that the BPEL variable Validate_CreditCardOut contains XPath expression message/part/Validate_CreditCardResponse/Validate_CreditCardResult with value equal to 1;
- Check that the XPath message/part/city does not exist in the BPEL variable OrderPartnerOperationIn;
- Assert that the Response time is maximum 5000 ms.

9) Execution of the test case and collection of test results by Test Case Generation and Execution Tool. For the current test case all obtained results are the same as the expected ones.

The definition of the assertions, the SOAP request template corresponding to the generated test case as well as the SOAP response from the business process are presented in Figure 2.

![Test Case Generation and Execution Tool](image)

Figure 2 Test Case "Missing City"

Table 2 shows all test cases executed to achieve full path coverage of “Check Payment” business process.

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Short description</th>
<th>Input Data</th>
<th>Expected Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has City</td>
<td>In this test case all provided data is correct and the user has filled its full address including postal code, city and state.</td>
<td>Email: <a href="mailto:john@gmail.com">john@gmail.com</a>&lt;br&gt;State: TX&lt;br&gt;Postal Code: 77590&lt;br&gt;Credit Card No: 4111111111111111&lt;br&gt;Total: 40&lt;br&gt;Currency Code: USD</td>
<td>HTTP Status Code = 200&lt;br&gt;Response Time &lt; 5000ms&lt;br&gt;Contains &lt;city&gt;&lt;state&gt;&lt;creditCardNumber&gt;&lt;total&gt;&lt;currencyCode&gt;Not Contains &lt;errorCode&gt;Invalid Credit Card Number&lt;errorCode&gt;</td>
</tr>
<tr>
<td>Missing City</td>
<td>In this test case all provided data is correct, but the user has not provided its city.</td>
<td>Email: <a href="mailto:john@gmail.com">john@gmail.com</a>&lt;br&gt;Postal Code: 77590&lt;br&gt;Credit Card No: 4111111111111111&lt;br&gt;Total: 40&lt;br&gt;Currency Code: USD</td>
<td>HTTP Status Code = 200&lt;br&gt;Response Time &lt; 5000ms&lt;br&gt;Contains &lt;city&gt;&lt;state&gt;&lt;creditCardNumber&gt;&lt;total&gt;&lt;currencyCode&gt;Not Contains &lt;errorCode&gt;Invalid Credit Card Number&lt;errorCode&gt;</td>
</tr>
<tr>
<td>Invalid Credit Card</td>
<td>In this test case the user has given a wrong credit card</td>
<td>Email: <a href="mailto:john@gmail.com">john@gmail.com</a>&lt;br&gt;State: TX&lt;br&gt;City: TEXAS</td>
<td>HTTP Status Code = 200&lt;br&gt;Response Time &lt; 5000ms&lt;br&gt;Contains &lt;errorCode&gt;Invalid Credit Card Number&lt;errorCode&gt;</td>
</tr>
</tbody>
</table>
Suppose that “Addresslookup” partner web service has limited number of invocation in test mode. This problem could be solved by using the approach for web service isolation, including following additional steps of the proposed methodology:

A1) Selection of web service to be isolated, namely “Addresslookup” web service.

A2) Formal description of the isolation. It is necessary to isolate the invoke activity “PostCodeInvoke” and assign values (e.g. "TEXAS CITY") to the returned variables. The configuration file for isolation of “Addresslookup” web service has the following form:

```
C:/CheckPayment/src/CheckPayment.bpel
---
process//invoke[@name='PostCodeInvoke']
  $OrderPartnerOperationOut.Order_Output/ns0:city='TEXAS CITY'
  $OrderPartnerOperationOut.Order_Output/ns0:state='TX'
---
```

A3) Transformation of the business process by Isolation Tool. The tool replaces the invoke activity calling the selected partner service with assign activities that simulate the expected message data by the business process.

The test cases described in Table 2 are executed on the transformed version of the business process. The obtained results show that the business process is not affected by the isolation procedure.

Suppose that one of the partner web services returns a response with some delay. In order to test the behaviour of the process in such situation, the following additional steps of the proposed methodology are executed:

B1) Formal description of the fault that need to be simulated through Fault Injection Tool. The tool provides a delay of 20 seconds for the response from the “ValidatorDemo” web service, formalized as follows:

```
C:/CheckPayment/src/CheckPayment.bpel
---
+proxy
  wait=20
  error_ratio=0
  http://mathertel.de/AJAXEngine/S03_AJAXControls/ValidatorDemo.asmx
  $ValidateEMailIn.parameters=$ValidateEMailOut.parameters
---
```

B2) Creation of transformed version of the BPEL process by the Injection Tool. In the transformed version of the process the communication is redirected to a Proxy service that simulates the defined fault.

The test cases described in Table 2 are executed again on the transformed version of the business process. The obtained results show that the Response time assertion is not satisfied. This is due to the simulated delay for the response from the “ValidatorDemo” web service.

The presented case study proves the feasibility of the proposed methodology. It shows how the methodology can be used for different types of testing, namely functional, performance and robustness testing. In addition, automation of the methodology’s steps is provided. This is achieved thorough usage of software tools that implement approaches for data dependency analysis, fault injection, web service isolation, and test case generation and execution.
CONCLUSIONS AND FUTURE WORK

The paper proposes a methodology for testing web service orchestrations, described with WS-BPEL, and provides results from its application in real testing process. The presented case study covers functional testing, where the main steps of the methodology are performed, as well as performance and robustness testing, including execution of the additional steps of the methodology. The obtained results prove the feasibility of the proposed methodology and its corresponding testing framework. The functional testing is achieved through generation and execution of several test cases providing full path coverage of the business process under test. The robustness and performance testing is achieved through isolation of the business process from one of its partner web services and simulation of fault causing message delay.

The future work includes validation of the proposed methodology in two directions: testing of more complex business processes and investigation on usage of other appropriate tools for execution of its steps. Thus, it will be possible to refine the methodology in order to define requirements to the software tools and platforms for SOA testing. Further on, the effectiveness of the proposed methodology will be explored in comparison with the existing ones.

REFERENCES

ACKNOWLEDGEMENT
This work is supported by the Scientific Research Fund of Sofia University under Grant No. 107/19.04.2013 and the National Scientific Fund of Ministry of Education, Youth and Science under agreement No. DOO2-182.

ABOUT THE AUTHORS
Denitsa Manova, Rila Solutions, Phone: +359 2 97096100, E-mail: denitsat@rila.bg.
Prof. Dr. Sylvia Ilieva, Assoc. Prof. Dr. Dessislava Petrova-Antonova, Department of Software Engineering, Sofia University, Phone: +359 2 9710400, E-mails: sylvia@acad.bg, d.petrova@fmi.uni-sofia.bg.