TASSA: A Testing as a Service Framework for Web Service Compositions

Dessislava Petrova-Antonova[1], Sylvia Ilieva[1,2] and Denitsa Manova[3]
[1]*Depatment of Software Engineering, Sofia University "St. Kliment Ohridski", Sofia, Bulgaria*
[2]*Institute of Information and communication technologies, BAS, Sofia, Bulgaria*
[3]*Rila Solutions, Sofia, Bulgaria*

Keywords: Cloud Computing, Service-Oriented Architecture, Testing-as-a-Service, Web Services, Web Service Compositions, WS-BPEL.

Abstract: Testing-as-a-Service (TaaS) is a new quality assurance model addressing the challenges of software testing in the cloud. The missing access to the hardware or different software configurations as well as the difficulties of building a test environment are examples for common problems in the testing process. This paper addresses such problems by proposing a TaaS-enabled framework offering testing services on as-needed basis. The framework, called Testing as a Service Software Architecture (TASSA), supports testing of web service compositions described with Business Process Execution Language for Web Services (WS-BPEL). Its core functionality includes fault injection and dependencies isolation of the application under test. It is implemented as web services deployed on cloud infrastructure. In addition, the TASSA Graphical User Interface (GUI) for test case design and execution is implemented as a plugin for Eclipse IDE. It could be accessed from a local computer or used for building a cloud test lab on a virtual machine. Sample business process from wine industry is used for proving the feasibility of TASSA framework.

1 INTRODUCTION

Nowadays, the cloud computing is one of the hot topics in software development. It provides a new way for building software applications known also as Software-as-a-Service (SaaS). The challenges and business opportunities that cloud computing brings affect all different activities of software engineering, including software testing. A new on-demand testing model, called Testing-as-a-Service (TaaS) became available.

In general, the software testing faces various difficulties due to lack of time and testing experience, limited resources and unclearly defined requirements and testing criteria. But, the most significant difficulties could appear before the beginning of the testing itself. The missing access to the hardware or different software configurations as well as building a test environment are examples for common problems surrounding the testing process of SaaS. However, with the emergence of cloud computing, the software testing gain new benefits represented by the TaaS:

- Access to virtual environments providing a variety software and hardware configurations;
- Possibility for deployment and/or usage different testing environments;
- Availability of on-demand testing services allowing testers to provision raw cloud resources at run time, when and where needed;
- Availability of multi-tenant testing services following given QoS requirements and Service Level Agreements (SLAs);
- Reduced cost of testing due to pay-per-use basis of testing services.

Following the current trends in software testing provided by TaaS, this paper proposes a framework, named TASSA, for testing web service compositions, called Testing as a Service Software Architecture (TASSA). The last three benefits are available in TASSA framework by implementation of its core functionality as cloud-based testing services.

Testing web service compositions is a challenging task, since their implementation follows the Service-Oriented Architecture (SOA). Although many research efforts are focused on SOA testing in the past few years, the following difficulties still remain

Petrova-Antonova, D., Ilieva, S. and Manova, D.
TASSA: A Testing as a Service Framework for Web Service Compositions.
DOI: 10.5220/0005844400330042
In *Proceedings of the International Workshop on domAin specific Model-based AppRoaches to vErificaTion and validaTiOn (AMARETTO 2016)*, pages 33-42
ISBN: 978-989-758-166-3

unsolved:

- Distributed and heterogeneous nature of SOA applications. Implementation of SOA applications requires composition of web services that are built and deployed on heterogeneous platforms. These web services are outside organization boundaries and are hard to be tested since they are owned by different stakeholders. 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 complicates the testing due to the necessity of emulation of the missing or unavailable web services. *TASSA framework addresses it through support of dependency isolation and fault injection of software under test from external web services.*
- Lack of knowledge about testing artefacts. When testing traditional applications the testers rely heavily on their GUIs. However, SOA testers miss such convenience since the web services expose programming interfaces defined with Web Service Description Language (WSDL). In addition, they do not have access to the design documents and source code of the integrated software components, which often decreases the testing efficiency. *Again, dependency isolation functionality of TASSA framework addresses this challenge.*
- Difficulties to reproduce testing environments. Typically, SOA solutions integrate products from different vendors following complex technical specifications and standards. That is why, it is difficult to test all software configurations and varying load on SOA infrastructure and underlying network. Thus, high technical competence is required from the testers and more attention on performance, robustness and security testing is needed. *The Graphical User Interface (GUI) of TASSA framework is fully integrated with Eclipse Integrated Development Environment (IDE). Thus, end-to-end testing environment for web service compositions provided.*
- Lack of full automation. Although various approaches and tools for web service composition 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, etc. *However, in order to perform efficient testing, it is important to integrate all testing activities in a common testing environment, which is the TASSA framework case.*

TASSA framework provides end-to-end testing environment for web service compositions, described with Business Process Execution Language for Web Services (WS-BPEL) that takes the benefits of the TaaS model. It consists of two main components:

- GUI for test case specification and execution that is available as a plugin for Eclipse IDE;
- TaaS functionality for fault injection and dependencies isolation that is available as web services deployed on a cloud infrastructure.

The rest of the papers is organized as follows. Section 2 outlines the related work. Section 3 is devoted to the architecture of TASSA framework. Section 4 presents the implementation of the TaaS functionality for fault injection and dependencies isolation. Section 5 describes a case study of testing sample business process with TASSA framework as a proof of concept. Section 6 concludes the paper and gives directions for future work.

2 RELATED WORK

The related work, presented in this section, approaches and frameworks following TaaS concept.

An extensive overview of recently proposed approaches and tools for functional, structural and security testing of web services is presented in (Bartolini et al., 2012). The authors of (Bucchiarone et al., 2007) survey the current solutions for testing web service compositions. The most work in these surveys are focused on web service signatures, namely WSDL descriptions (Bartolini et al., 2008; Dong, 2009; Noikajana and Suwannasart, 2009; Bai et al., 2005, Lopez et al., 20013; Masood et al., 20013). However, WSDL interface does not provide a semantic information for web services and a behavioral description of them, which is important when testing web service orchestrations. In contrast, TASSA framework is focused on testing of web service compositions, described with WS-BPEL. This approach of SOA testing is adopted by several works. In order to perform control and data flow testing, the authors of (García-Fanjul et al., 2006) propose the use of model checkers for test cases generation from BPEL descriptions. Other approaches (Hou et al., 2009; Yuan et al., 2006; Karam et al., 2007; Li et al., 2008) focus on analysis of test paths derived from graph models representing the composition specification. In this direction, Yuan et al. (2006) propose a graph-search based approach transforming the BPEL into an extension of a control flow graph and generating test data for each path by using constraint solvers. Other approaches, e.g. (Cao et al., 2009), propose online testing algorithms for web services composition using BPEL.

Currently, the TaaS benefits focusses the attention of both industry and academic communities on building cloud-based solutions for software testing. Recently, a considerable number of definitions for TaaS were proposed. Each of them emphasizes on different TaaS perspectives. According to (Candea et al, 2010) TaaS implies two ideas: first, providing software testing as a web service that is competitive and easily accessible, and second, performing automated testing using the huge, elastic resources of cloud infrastructure. TaaS is viewed as a cloud-based service that automates the software testing in (Parveen and Tilley, 2010). The migration of software testing to the cloud is presented from the following points of view – the characteristics of the software under test and the type of testing performed on the software. As pointed in (Yu et al., 2010) TaaS is "a new model to provide testing capabilities to end users". A more thorough definition of TaaS is provided in (Gao et al., 2013). On one hand, TaaS is explained as a service model for software testing available on-demand. On the other hand, TaaS is described as a new business model for software testing providing cost-sharing and cost-reduction due to its pay-as-you-test abilities.

A framework of TaaS as a new model to improve the efficiency of software testing is proposed in (Yu et al., 2010, 2009). It consists of four layers: Test Service Tenant and Contribution layer, Testing Service Bus layer, Testing Service Composite layer, and Testing Service Pool layer. The idea of the framework is similar to the one of Universal Description Discovery and Integration (UDDI) registry. Its main functionality includes registering, matching, reasoning, classifying and scheduling of testing services in order to provide TaaS-based service compositions to the end users. The authors of (Zhu and Zhang, 2012) apply similar approach by proposing a framework for collaborative testing of web services. The framework uses various test services that interoperate to complete the testing tasks. They are registered, discovered, and invoked at runtime in order to achieve testing on-the-fly with a high degree of automation. Prescriptions for implementation of TaaS strategy by the software organizations are introduced in (Sathe and Kulkarni, 2013).

Although there are a number of recently published research papers addressing TaaS issues, challenges, and needs, there is a very few published papers focusing on web service composition testing. In (Yan et al., 2012) requirements for web service load testing are identified and a WS-TaaS platform for such type of testing is pro-posed. The platform is based on an existing Cloud PaaS platform, called Ser-vice4All. Unfortunately, it does not support testing of web service compositions and its functionality is limited to that provided by the Apache JMeter (2015). A cloud platform for testing Service-Oriented Architecture (SOA) orchestrations, called MIDAS, is proposed in (Herbold et al., 2015). The MIDAS platform adopts SOA paradigm, so all its functionality is exposed as services deployed on a cloud infrastructure. The supported types of testing are functional testing, security testing and usage-based testing. A limitation of the MIDAS platform is that it allows testing of service interactions with SOAP messages. The test methods require specification of input models using UML-based language, called MIDAS DSL. This is a drawback of the MIDAS platform, since usually the SOA orchestrations are described with languages such as WS-BPEL, Business Process Modelling Notation (BPMN) and Windows Workflow Foundation (WWF).

There are several cloud-based commercial platforms on the market providing TaaS. SOASTA CloudTest (2015) and IBM Rational Performance Tester (2015) are solutions for load and performance testing. Sauce Labs is a platform for testing mobile and web applications (2015). It allows testers to create test manually or using Appium, Selenium or JavaScript unit. Oracle provides a platform covering the testing process end-to-end (2015). It automates the provisioning of so called test labs, which includes the application under test and the software tools for functional and load testing. The most powerful solution for cloud testing is provided by Parasoft (2015). Their testing platform is designed to support functional, performance, load and security testing of all the protocols and technologies that make cloud-based applications possible (HTTP/S, JMS, MQ, ESB, PoX, JDBC, RMI, Tibco, SMTP, .NET WCF, SOAP/WSDL, REST/WADL, etc.). In addition, the behavior of dependent applications (third-party services, mainframes, database, etc.) can be emulated using Parasoft's service virtualization.

The commercial cloud-based solutions presented above are focused on building test labs mainly by installing the currently provided testing tools on cloud infrastructure. Most of them do not provide support for testing of web service compositions, which is the main purpose of TASSA framework. In addition, TASSA framework follows SOA paradigm similar to MIDAS platform and exposes its core functionality as a web services deployed on the cloud. Its GUI could be accessed from a local computer or used for building a cloud test lab on a virtual machine.

3 ARCHITECTURE OF TASSA FRAMEWORK

TASSA is a cloud-based framework for testing web services orchestrations, de-scribed with WS-BPEL. It reduces the testing effort by providing functionality for automation and tracing of testing steps performed during test project lifecycle.

The high level architecture of TASSA framework is presented in Fig. 1. It consists of two main components:

- Graphical User Interface (GUI) for test case specification and execution;
- TaaS functionality for fault injection and dependencies isolation.

The GUI provides functionality that is separated in three layers: Test template design; Test case generation; and Test case execution. It is implemented as a plugin for Eclipse IDE.

At test template design layer a version of the business process under test, called template, is created by transformations over original *.bpel file. Two types of transformations are supported:

- Isolation of activity – isolates the business process from its external dependencies by replacing one or more activities with such ones that mimic their output.
- Fault injection – injects delays, errors, etc. in the message exchange for a particular activity.

The isolation is performed through invocation of appropriate operation of Simulate web service, while the fault injection is provided by the *ProxyInvoke* web service. Both web services are deployed on an application server using Amazon EC2. Thus, the time to obtain and boot a new server instances is reduced allowing capacity to be scaled quickly as the computing requirements change. The deployment model of TASSA framework is shown in Fig. 2.

When a new test project is created a default read only template, called *Original*, is generated. It holds the original *.bpel file and all files from which its deployment and execution depend on. It can be used as a base for creation of new test templates or test cases. When a new test template is created, a folder structure is associated with it. It contains the following items:

- **Dependencies folder**, which contains all files the business process's deployment and execution depend on.
- **deploy.xml file**, which is the deployment descriptor for the Apache ODE Server;
- ***.tm file**, which represents the created test template and describes the actions, called steps, that are applied to the business process;
- ***.bpel file**, which corresponds to a transformed business process that will be used during test execution.

At the next layer, the test cases are generated automatically from the test tem-plates. The test assertions can be specified manually by the tester in two ways:

- Defining an *XPath expression* over a particular business process's variable that will be evaluated during test case execution;
- Directly editing fields of particular business process's variable using provided XML editor.

The test assertions are useful in order to validate the response messages re-turned by the partner web services. They are stored in XML file with root element, called *ListOfAssertions*. Each assertion starts with *Assert* element that has three child elements: *variable*, *XPath* and *document*. The variable element of the assertion keeps the name of the business process's variable for which the test assertion is defined. The *xPath* element contains an *XPath expression*, if such is defined. If the test assertion is specified by directly editing of the business process's variable, the document element is filled with the content of that variable.

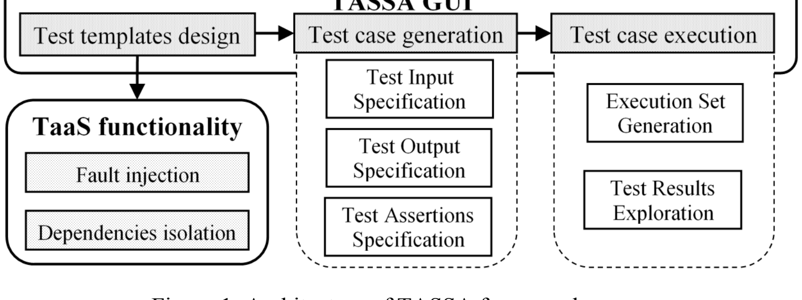


Figure 1: Architecture of TASSA framework.

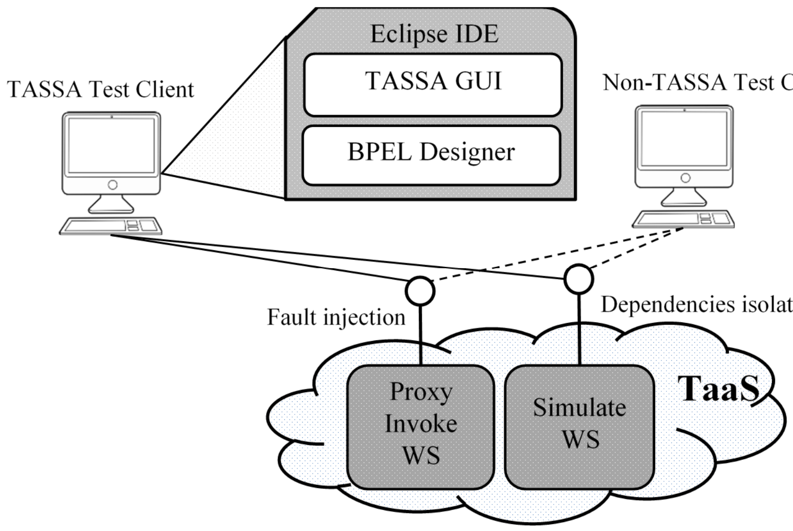


Figure 2: Deployment model of TASSA framework.

The test cases need to be added to execution set in order to be executed. The execution sets allow test cases to be grouped for simultaneous execution. The result from execution of each test case is stored in XML document, which elements are following:

- **testCaseName** – name of the test case;
- **request** – request to the business process;
- **response** – response from the business process;
- **executionTime** – duration time of the test case;
- **compareResult** – test verdict: true, if the test is passed, and false – otherwise;
- **traceEnabled** – indication for tracing ability activation;
- **activitiesPassed** – activities that have been called during BPEL process execution;
- asserts – test case assertions;
- **executionDate** – execution date.

4 CASE STUDY

This section presents a proof of concept of TASSA framework through a case study using a business process, called *Grapes Order*. The business process serves by wine companies while deciding to buy grapes.

4.1 Business Process under Test

The Grapes Order is a synchronous business process that calls three partner web services, namely *Grape Producer North*, *Grape Producer South* and *Perform Order*. Its graphical view is shown in Fig. 3. It takes as an input information about the grapes variety, the quantity and the delivery address. Then the *Grape Producer North* and *Grape Producer South* partner web services are invoked in parallel flow to check the price of the grapes. After that an order is placed in the inventory with the cheaper grape, again using one of the partner web services. Finally, *Perform Order* partner web service finalizes the order by calculating the total price and the expected delivery date.

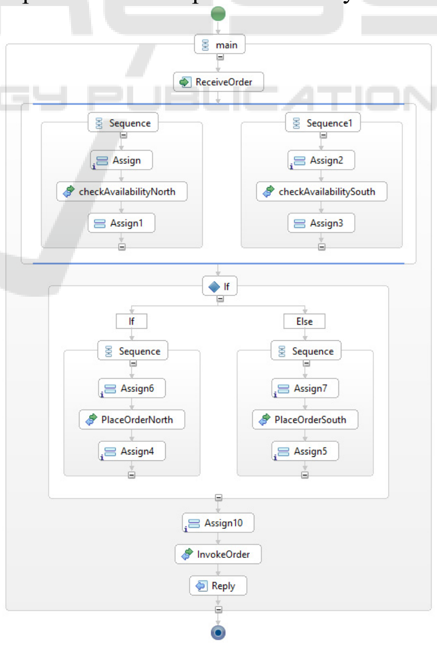


Figure 3: Wine Producer Business Process.

4.2 Test Template Design

The test template includes a version of the business processes described with the BPEL language and all accompanied documents like WSDL descriptions, XSD schemas, etc. Using TASSA framework it is possible to transform the original BPEL file to isolate the business process from its external dependencies or to simulate faults. Each transformation actually produces a valid BPEL file that imitates the behavior of the initial one in testing conditions.

In order to test the business process workflow, the dependencies from the partner web services need to be removed. The *Grapes Order* business process has three partner web services and five invokes to them. At isolation an *Invoke* activity should be selected and then substitution values should be specified. As a result, the *Simulate* web service of TASSA framework transforms the BPEL file so that the partner web service invocation is removed. Fig. 4 shows the BPEL code of the invoke activity that calls the *checkAvailabilityNorth* operation of *Grape Producer North* partner web service, while Fig. 5 presents its transformation.

```
<bpel:invoke name="checkAvailabilityNorth"
  partnerLink="GrapeProducerNorth"
  operation="checkAvailability"
  portType="ns:GrapesProducerNorth"
  inputVariable="GrapeProducerNorthRequest"
 outputVariable="GrapeProducerNorthResponse">
</bpel:invoke>
```

Figure 4: Original “checkAvailabilityNorth” activity.

```
<bpel:assign
name="AssignIsolate1checkAvailabilityNorth">
 <bpel:copy>
  <bpel:from>
   <bpel:literal xml:space="preserve">
    <Q1:checkAvailabilityResponseElement
             xmlns:Q1="…" xmlns:xsi="…">
    <Q1:isAvailable>true</Q1:isAvailable>
    <Q1:Available_Quantity>1.0
             </Q1:Available_Quantity>
    <Q1:Price>13.4</Q1:Price>
    <Q1:Delivery_Time>48</Q1:Delivery_Time>
    </Q1:checkAvailabilityResponseElement>
   </bpel:literal>
  </bpel:from>
 <bpel:to part="parameters"
variable="GrapeProducerNorthResponse"/>
 </bpel:copy>
</bpel:assign>
```

Figure 5: Transformed “checkAvailabilityNorth” activity.

Similar transformations are performed regarding *checkAvailabilitySouth* operation of *Grape Producer South*, *PlaceOrderNorth* operation of Perform Order partner web service, *PlaceOrderSouth* operation of *Perform Order* partner web service and *InvokeOrder* operation of *Perform Order* partner web service.

TASSA framework supports robustness testing providing functionality for an-other transformation. In such case the BPEL file is transformed so that the call to a partner web service is replaced with a call to a *ProxyInvoke* web service. Thus, the fault injection is performed.

The Grapes order business process is injected with four type of faults supported by TASSA framework.

The simulation of delay in the response from *Grape Producer South* web service is performed by replacement of corresponding *Invoke* activity with two *Assign* activities and one new *Invoke* activity. The first *Assign* activity provides configuration information to the *ProxyInvoke* web service as follows:

- **Wait interval** – an integer value that defines the delay of message in seconds;
- **Error factor** – an integer value that that defines the kind of error will be injected (1÷100: insert random errors in the data, which would possible break the XML structure; 0: usually used with Wait interval to delay the message; - 1: replace the original values in the message; -2: interrupt the message)
- **End point address** – an end point address of the partner web service;
- **Activity variable** – input variable of the partner web service, which invocation is injected with faults.

The second *Assign* activity copies the result from invocation of *ProxyInvoke* web service to the output variable of the partner web service, which invocation is injected with faults. The Invoke activity calls *ProxyInvoke* web service.

Similar transformations are performed regarding *Grape Producer North* and *Perform Order* partner web services.

Test templates created for the business process under test are listed in Table 1.

TASSA framework also supports generation of test templates from the existing one. Thus, a rollback to the previous version of the business process under test could be performed.

4.3 Test Case Definition

Test cases created with TASSA framework are in correlation with test templates. Each test case is linked to exactly one test template. It consists of test in-put, expected output and assertions if any. During test case execution the BPEL file from the test template is deployed on the application server, the test

Table 1: Test templates (TT).

TT	Description
TT1	Original business process without any transformation
TTI1	Isolation of all partner web services
TTI2	Isolation of Perform Order partner web service
TTI3	Isolation of Grape Produces North partner web service
TTI4	Isolation of Grape Produces South partner web service
TTF1	Simulation of small delay in the response from partner web service
TTF2	Simulation of large delay in the response from partner web service
TTF3	Simulation of missing partner web service
TTF4	Simulation of low level noise in the response from partner web service
TTF5	Simulation of high level noise in the response from partner web service
TTF6	Simulation of response with wrong data from partner web service

input from the test case is sent as a request to the business process, its response is compared to the expected output and the assertions from the test case are checked.

Actually, many test cases could be created from the same test template. Thus, several requests with different test inputs will be send to the same version of the business process (deployed BPEL file). Using this feature of TASSA framework a set of test cases are created from the test templates described above.

Two test cases for the test templates that isolate *Grape Producer North* and the *Grape Producer South* web services are created (TCI1 and TCI2). They provide full path coverage of the business process since both *True* and *False* branches of the *If* activity are executed. The same test scenarios are designed when the partner web services are available and their operations are actually invoked (TC1 and TC2).

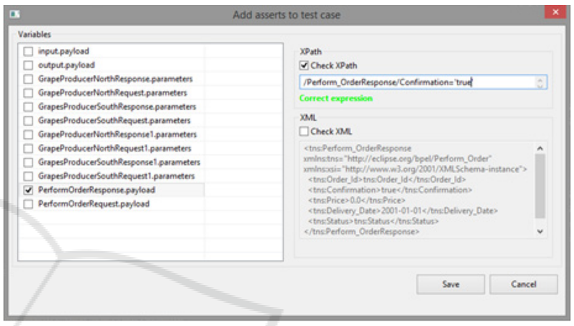
Several test cases with invalid data are also created (negative test cases):

- TCN1 – send request with zero quantity;
- TCN2 – send request with quantity over availability;
- TCN3 – send request with invalid grape type;
- TCN4 – send request with invalid quantity;
- TCN5 – send request with invalid delivery type.

In order to perform robustness testing the following test cases are defined using fault injection features of TASSA framework:

- TCF1 – simulates a small wait interval;
- TCF2 – simulates a big wait interval;
- TCF3 – simulates missing partner web service;
- TCF4 – simulates low level noise in the response from partner web service;
- TCF5 – simulates high level noise from partner web service;
- TCF6 – simulates response from partner web service with random data.

As it was already mentioned, TASSA testing framework supports specification of test assertions, namely XML assertions and XPath assertions. Fig. 6 shows a sample assertion defined for the response from Perform Order partner web service.



Variables:
- [ ] input.payload
- [ ] output.payload
- [ ] GrapeProducerNorthResponse.parameters
- [ ] GrapeProducerNorthRequest.parameters
- [ ] GrapesProducerSouthResponse.parameters
- [ ] GrapesProducerSouthRequest.parameters
- [ ] GrapeProducerNorthResponse1.parameters
- [ ] GrapeProducerNorthRequest1.parameters
- [ ] GrapesProducerSouthResponse1.parameters
- [ ] GrapesProducerSouthRequest1.parameters
- [✓] PerformOrderResponse.payload
- [ ] PerformOrderRequest.payload

XPath: [✓] Check XPath

```
/Perform_OrderResponse/Confirmation='true'
```

Correct expression

XML: [ ] Check XML

```
<tns:Perform_OrderResponse
xmlns:tns="http://eclipse.org/bpel/Perform_Order"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <tns:Order_Id>tns:Order_Id</tns:Order_Id>
  <tns:Confirmation>true</tns:Confirmation>
  <tns:Price>0.0</tns:Price>
  <tns:Delivery_Date>2001-01-01</tns:Delivery_Date>
  <tns:Status>tns:Status</tns:Status>
</tns:Perform_OrderResponse>
```

Figure 6: GUI for writing an assertion.

XML Assertion compare the specified XML message with the received one. In many cases, especially when dynamic data such as IDs or Dates are used, it is better to check only part of the XML message. In such cases the XPath assertion is recommended to be used.

4.4 Test Case Execution

Test cases are grouped in *Execution sets*. The execution set is a list of test cases, which are executed in the order specified in the list. The test cases in the execution set can be logically grouped. For example all negative test could be arranged in one execution set and all tests performing isolation could be arranged in other execution set. This feature of TASSA framework is especially useful when it is applied to test cases that should be executed in proper order. For ex-ample, one may need to execute a test which adds some quantity for a given item and then to execute a test in which this item is sold. Similarly to the test tem-plates, the test cases are reusable. Single test case can be placed in more than one execution set.

When execution sets are ready to run on the application server, for each test case the results are collected and the assertions are checked. The results are written in log files that are grouped according to the execution set they belong to.

Table 2 shows the results from execution of the test cases when partner web services are isolated and when partner web services are actually invoked and the business process receives valid test data. Using TASSA framework the isolation is performed in a way that the business process acts in the same way as when the real partner services are available and called.

Table 2: Test cases showing isolation of partner web services.

Test Case	Input	Expected Output	Received Output
TCI1	white,1,fast	reserved, north,48h	reserved, north,48h
TCI2	red,2,normal	reserved, south,48h	reserved, south,48h
TC1	white,1,fast	reserved, north,48h	reserved, north,48h
TC2	red,2,normal	reserved, south,72h	reserved, south,72h

Table 3 shows the results from execution of test cases when the business process receives invalid data (negative test cases). Usually a well formed business process should catch exceptions and send proper messages when incorrect action is performed. That is why, an informative or error message is expected to be received in case of testing with invalid values. After performing the negative tests (those with invalid test data) and checking the execution logs, it was found that the business process does not catch several exceptions.

Table 3: Test cases with invalid data.

Test Case	Input	Expected Output	Received Output
TCN1	white,0,fast	informative message	reserved, north,48h
TCN2	red,99999,normal	informative message	log error
TCN3	123,1,fast	informative message	log error
TCN4	red,A,normal	error message	log error
TCN5	red,1,123	informative message	log error

Table 4 shows the results from robustness testing of the business process. Such testing suppose that the system under test should not crash and respond with error message, overcoming violations if it is possible.

The *Grapes Order* business process acts properly in case of small delays of the response from a partner web service. It returns an expected output with a delay specified with the wait interval parameter of the Proxy Invoke web service of the TASSA framework. In case of long message delay from partner web service, missing partner web service or noise in the communication channel the business process crashes and the server logs should be explored in order to fix the problems. Test results obtained when random invalid data is injected in the response form partner web service show differences between the expected and the received outputs. It is expected that when a random invalid data is sent, the process should respond with informative or error message. As Table 4 shows the business process accepts such data and responds with a common output. Therefore, additional fixtures should be done in the business process.

Table 4: Test cases performing robustness testing.

Test Case	Failure Parameter	Expected Output	Received Output
TCF1	Wait=10s	delayed common output	delayed common output
TCF2	Wait=20m	Time elapsed	Time elapsed
TCF3	Interrupt	error message	error message
TCF4	Noise range=1%	log error/common output/ informative msg	log error
TCF5	Noise range=60%	log error	log error
TCF6	Random	informative message/ error output	common output

5 CONCLUSIONS

The paper presents TASSA framework for testing web service compositions de-scribed with WS-BPEL. Its core functionality is implemented as web services deployed on a cloud infrastructure. A case study over a business process serving a wine company is used as a proof of concept. It shows the usefulness and benefits of TASSA framework as follows:

- Ability to test web service compositions even if partner web services are unavailable or under development through provided functionality for dependency isolation;
- Ability to perform robustness testing through

provided functionality for fault injection;

- Availability of on-demand testing services allowing testers to provision raw cloud resources at run time, when and where needed;
- Reduced cost of testing due to pay-per-use basis of testing services;
- Shows SUT's problems (bugs).

The future work includes application of TASSA framework to testing more complex business processes having tens partner web services. Its performance and efficiency will be evaluated in comparison with another automated testing tools as well as manual testing. Additional functionality for load testing and runtime monitoring is planned to be implemented.

ACKNOWLEDGEMENTS

The authors acknowledge the financial support by the Scientific Fund of Sofia University.

REFERENCES

Candea G., S. Bucur, Cristian Z, 2010. Automated Software Testing as a Service (TaaS). In *Proceedings of the 1st ACM Symposium on Cloud Computing*. pp. 155-160.

Parveen T., Tilley S., 2010. When to Migrate Software Testing to the Cloud?. In *Third International Conference on Software Testing*. Verification, and Validation Workshops (ICSTW), pp.424-427.

Yu L., W. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, Zhao W., 2010. Testing as a Service over Cloud. In *Proceedings of the Fifth IEEE International Symposium on Service Oriented Sys-tem Engineering*. pp. 181-188.

Gao J., X. Bai, Tsai W., 2013. Testing as a Service (TaaS) on Clouds. In *Proceedings of the Seventh IEEE International Symposium on Service-Oriented System Engineering*. pp. 212-222.

Yu L., L. Zhang, Xiang H., Su Y., Zhao W., Zhu J., 2009. A Framework of Testing as a Service. In *Proceeding of the Conference of Information System Management*.

Sathe A., Kulkarni R., 2013. Study of testing as a service (TaaS) – cost effective framework for TaaS in cloud environment. In *International Journal of Application or Innovation in Engineering & Management* (IJAIEM), Volume 2, Issue 5, pp.239-243.

Yan M, Sun H., Wang X., Liu X., 2012. Building a TaaS Platform for Web Service Load Testing. In *Proceeding of the IEEE International Conference on Cluster Computing*. pp. 576-579.

Apache JMeter, http://jmeter.apache.org/, last accessed on 11.06.2015.

Herbold S. et al., 2015. The MIDAS Cloud Platform for Testing SOA Applications. In *Proceedings of the IEEE 8th International Conference on Software Testing, Verification and Validation* (ICST), pp. 1-8.

SOASTA CloudTest, https://www.soasta.com/wp-content/uploads/2015/05/CT-Data-Sheet.pdf, last accessed on 12.06.2015.

SOASTA CloudTest, https://saucelabs.com/downloads/one_pager_sales_sheet.pdf, last accessed on 12.06.2015.

IBM Rational Performance Tester, https://www.ibm.com/developerworks/cloud/library/cl-loadtest-softlayer-trs/, last ac-cessed on 12.06.2015.

Oracle Testing as a Service, http://www.oracle.com/technetwork/oem/cloud-mgmt/ds-oracletesting-as-a-service-1905796.pdf, last accessed on 12.06.2015.

Parasoft Cloud Testing, http://www.parasoft.com/capability/cloud-testing/, last accessed on 12.06.2015.

Amazon EC2, http://aws.amazon.com/ec2/, last accessed on 18.06.2015.

Bartolini, C., Bertolino, A., Lonetti, F. and Marchetti, E., 2012. Approaches to functional, structural and security SOA testing. In *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, Valeria Cardellini, Emiliano Casalicchio, Kalinka Regina Lucas Jaquie Castelo Branco, Julio Cezar Estrella, Francisco Josè Mona-co (eds.). Hershey, PA, USA: IGI Global, pp. 381-401.

Bucchiarone, A.; Melgratti, H., Severoni, F., 2007. Testing Service Composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering* (ASSE).

Bartolini, C., Bertolino, A., Marchetti, E., Polini, A., 2008. Towards Automated WSDL Based Testing of Web Services. In *Proceedings of the 6th International Conference on Service-Oriented Computing*. Volume 5364 of LNCS, pp. 524-529.

Dong, W., 2009. Testing WSDL_based Web service automatically. In *Proceedings of the WRI World Congress on Software Engineering*. pp. 521-5.

Noikajana, S., Suwannasart, T. An improved test case generation method for Web service testing from WSDL-S and OCL with pair-wise testing technique. Proceeding of the 33rd Annual IEEE International Computer Software and Applications Conference, 2009, p 115-23.

Bai, X., Dong, W., Tsai, W.-T., Chen, Y., 2005. WSDL-based automatic test case generation for Web Services testing. In *Proceedings of the IEEE International Workshop on Service-Oriented System Engineering*. pp. 215-220.

Lopez, M., Ferreiro, H., Francisco, M.A., Castro, L.M., 2013. Automatic Generation of Test Models for Web Services Using WSDL and OCL. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC)*. pp. 483-490.

Masood, T., Nadeem, A., Ali, S., 2013. An automated approach to regression testing of Web services based on WSDL operation changes. In *Proceeding of the IEEE 9th International Conference on Emerging Technologies (ICET)*. pp. 1-5.

García-Fanjul J., J. Tuya, de la Riva Cl., 2006. Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN. In *International Workshop on Web Services Modelling and Testing*. pp. 83-94.

Hou, S.-S., Zhang, L., Lan, Q., Mei, H. Sun, J.-S., 2009 Generating effective test sequences for BPEL testing. In *Proceeding of QSIC'09*. pp. 331-340.

Yuan, Y., Li, Z. Sun, W., 2006. A graph-search based approach to BPEL4WS test generation. In *Proceeding of ICSEA*. pp. 14-22.

Cao, T.-D., Felix, P., Castanet, R. Berrada, I., 2010 Online testing framework for web services. In *Proceeding of ICST*. pp. 363-372.

Karam M., H. Safa, H. Artail., 2007. An abstract workflow-based framework for testing composed web services. In *Proc. of Int. Conf. on Computer Systems and Applications*. pp. 901–908.

Li Z. J., Tan H. F., Liu H. H., Zhu J., Mitsumori N. M., 2008. Business-process-driven gray-box SOA testing. In *IBM Systems Journal* 47. pp. 457-472.

Zhu H. Zhang Y., 2012. Collaborative Testing of Web Services. In *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 116-130.