ISBN 978-83-60810-51-4

Towards a Methodology for Testing of Business Processes

Sylvia Ilieva
Sofia University, Department of Software engineering, 125 Tsarigradsko shosse Blvd., Block 2
1113 Sofia, Bulgaria
Email: sylvia@acad.bg

Ilina Manova
Rila Solutions, Acad. G. Bonchev St., Building 27, 1113 Sofia, Bulgaria
Email: ilinam@rila.bg

Dessislava Petrova-Antonova
Sofia University, Department of Software engineering, 125 Tsarigradsko shosse Blvd., Block 2
1113 Sofia, Bulgaria
Email: d.petrova@fmi.uni-sofia.bg

Abstract—**Business processes naturally integrate web services implemented with different languages and technologies and executed in heterogeneous environment. Usually the integrated web services and their underlying infrastructure used to exchange messages are not under the control of process architects. This reflects the development and specifically complicates the testing. In this paper a methodology for testing of business processes is presented, which aims to enable automatic test case generation for path coverage functional testing, as well as to provide fault injection mechanisms for negative functional testing. The methodology is supported by a testing framework, called TASSA, that consists of several tools for design time testing of business process described according Web Service Business Process Execution Language (WS-BPEL) standard. The framework follows the Service-Oriented Architecture (SOA) principles and is validated through sample business process scenarios.**

I. Introduction

SERVICE-ORIENTED Architectures (SOA) allows software applications to interoperate in a new way in distributed environment. Currently, web services are the most widely adopted technology for implementation of SOA. In order to achieve a particular business objective, they are composed in complex business processes following Web Service Business Process Execution Language (WS-BPEL) standard [1].

Testing business processes brings several challenges due to the following reasons:

- Missing of graphical user interface of business process;
- Invocation of services which are external for business process under test;
- Need of additional efforts and tools for testing third party/external web services of the business process in order to validate their quality;
- Possibility for usage of particular web service by multiple business processes;
- Security issues established in distributed environment such as authentication, authorization, data integrity and privacy, etc.

The authors acknowledge the financial support by the National Scientific Fund, Bulgarian Ministry of education, youth and science, grant agreement No. DO02-182.

Implementation of business processes requires composition of web services that are built and deployed on heterogeneous platforms. These services are outside organization boundaries and 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 complicates the testing of the business processes due to the necessity of emulation of the missing or unavailable web services. Additional efforts are needed for generation of appropriate message data, which will replace the actual ones expected by the business process.

In order to address the above issues, the paper proposes a methodology for testing of business processes. The methodology covers the following testing activities: (1) isolation of the business process from its partner web services, (2) fault injection, and (3) test case generation and execution. These activities are automated through implementation of several tools, which are integrated in a common framework, called TASSA providing end-to-end testing of business processes described with BPEL.

The rest of the paper is organized as follows. Section II describes the proposed methodology. Section III presents TASA tools. Section IV is dedicated to the validation of the methodology through sample business processes. Section V gives brief description of the current BPEL testing approaches and SOA testing methodologies. Section VI concludes the paper.

II. TASSA Methodology

This Section describes the methodology that TASSA framework implements. As was mentioned in the previous Section the methodology covers three main approaches: (1) Isolation of the business process from external partner web services; (2) Injection of faults in the business process or communication channel leading to unexpected behavior of the process; and (3) Test case generation and management. Each of them consists of three common steps:

1. Formal description of the business process with BPEL

TASSA methodology proposes formal description of the business following WS-BPEL standard. Thus the logic of the business process including sequences of events and activities, state transitions and invocations of partner web services becomes more comprehensible and traceable, which in turn facilitates test case generation. This is due to the fact that the core of the standard relies on the BPEL language, which is XML based language having similar features to those of imperative programming languages such as variables, loops, branches, exception handling, etc.

2. Execution of set of activities in order to achieve the specific goal of the approach

This step is specific for each approach and will be described in detail in the next subsections of the paper.

3. Business process deployment

In order to be tested the transformed business process need to be deployed on a suitable application server, which in case of BPEL description of the process could be JBoss, GlassFish, WebSphere and so on.

A. Isolation of the business process

The isolation of the business process from its partner web services removes the external dependencies of the process. Thus the business process could be tested even if particular web service is not available or is not still developed.

The isolation of the business process under test includes the following steps:

A.1. Identification of the web services that should be isolated

This step includes identification of the web service operations that should be simulated and the corresponding messages exchanged during web service invocation. Next, appropriate data are generated in order to replace the messages expected by the process.

A.2. Preparation for execution of isolation

This step requires formal description of the data generated on the previous step in order to be used as parameter that will be passed to a particular tool performing isolation activity. It includes definition of a message for invocation of appropriate TASSA tool.

A.3. Business process transformation

This step produces transformed version of the business process, in which the targeted web services are isolated. It is performed according to the transformation rules defined on the step 1.

After completion of the steps presented above the business process can be executed without availability of its partner web services. Thus it allows to be tested even if some of its building blocks are under development. The overall time for development will be reduced due to ability for testing of the business process in parallel with its partner web services.

B. Fault injection

The goal of the fault injection is to simulate faults during message exchange of the business process and its partner web services in order to generate negative test cases. Currently covered by the methodology situations that could be simulated are (1) overload of the communication channel that leads to delay of sending or receiving a message, (2) failure of the communication channel that leads to impossibility of sending or receiving a message, (3) noise in communication channel that leads to receiving a message with syntax and structure errors, and (4) wrong business logic of particular web service that leads to sending or receiving a message with syntax errors in its data.

The fault injection of the business process includes the following steps:

B.1. Description of faults that should be injected

This step requires description of the faults that will be simulated and their parameters. The fault description includes identification of the message exchanged when a failure is expected to occur, modification of the communication channel and the activity corresponding to the identified message. It could be executed manually or using TASSA framework.

B.2. Preparation for execution of injection

The step includes formal description of the faults as parameters that will be passes to a particular tool performing injection.. It includes preparation of a message for invocation of appropriate TASSA tool.

B.3. Business process transformation

This step produces transformed version of the business process, in which faults are injected. It is performed according to the transformation rules defined on the previous steps.

C. Test case generation and execution

Test case generation of the business process includes the following steps:

C.1. Data dependency analysis of the business process

Data dependencies of the business process are analyzed in order to find different execution paths. Possible solution is to transform the business process into tree structure that present the execution paths according to the values of the business process's variables.

C.2. Path selection

On this step a particular execution path of the business process is selected. The variables from which the selected path depends on are also identified.

C.3. Test data generation

This step requires generation of value for all variables identified on the selected path. The values should be chosen so that the business process to proceed on the desired path.

C.4. Isolation of the business process

This step produces transformed version of the business process, in which all variables from which the selected path depends on are replaced with their corresponding values generated on the previous step.

III. TASSA TOOLS

The implementation of the TASSA framework follows the methodology described in Section II. Several tools were produced that are described in this Section. Since the BPEL is widely accepted language for description of business processes, TASSA tools are implemented based on WS-BPEL standard and are integrated in NetBeans environment.

A. Isolation tool

The Isolation Tool (IT) provides temporary removal of BPEL process dependencies from one or more external web services. This allows the tester to control the returned results of web services and pre-determine the possible routines in the BPEL process, as well as to continue testing even if a particular web service is missing.

The BPEL process's dependency upon partner services can be described as follows:

- synchronous execution of operation provided by an external service (Invoke activity in the BPEL process description);
- asynchronous execution of operation provided by an external Service (combination of Invoke and Receive activity in the BPEL process description);
- unforced message receipt from external service (Pick activity);
- sending message to external service (resulting from an ingoing message);
- HumanTask activity, which requires human intervention and which affects the application through its output data (operator-entered values).

To eliminate the dependency upon Invoke activity the following actions should be conducted:

- Modification of the process, where the relevant Invoke activity is replaced with Assign activity to assign a specific values to the output variable;
- When isolating the process from an activity a test artifact should be created – a variant of the BPEL process, in which the Invoke activity is replaced by an Assign activity.

The other dependencies are handled in a similar way.

B. Fault injection tool

The main task of Fault Injection Tool (FIT) is to simulate faults during message exchange in order to generate negative test cases. The possible situations that are simulated are described in Section II. FIT takes as input a BPEL process under test, a list with failure parameters and a string with values, which correspond to the arguments of the activity causing the failure. It returns a transformed BPEL process with simulated failure.

The fault injection process consists of the following steps:

- identification of message exchanged when the failure is simulated;
- modification of communication channel, so that the failures expected by the tester occur;
- modification of an activity that corresponds to the message in order to send message to the proxy created between the message sender and receiver;
- serialization of input arguments of the real receiver (marshalling);
- invocation of the proxy;
- deserialization of output arguments and sending to the real receiver (unmarshalling).
- Similar steps are performed for the response of the invocation.

C. Value generation tool

The goal of Value Generation Tool (VGT) is to generate valid values for all field of a given variable defined with XML Schema Definition (XSD) in the BPEL process. Currently the main functionality of VGT is provided by a tool, called WS-TAXI, which is developed by a research team of Software Engineering Research Laboratory at the ISTI (Istituto di Scienza e Tecnologie dell'Informazione) in Pisa. WS-TAXI generates compliant XML instances from a given XML Schema by using well-known Category Partition technique. VGT takes as input a BPEL process under test and an array with identifiers of variables, whose values need to be generated.

D. Data dependency analysis tool

Data Dependency Analysis Tool (DDAT) produces a list of variables for a given execution path in the BPEL process. It receives as input a BPEL process and an array of unique identifiers of activities, describing the path that the BPEL process needs to follow.

During data dependency analysis three operations implemented in the TASSA framework are invoked: Analyze, Emulate and Apply. The operation Analyze provided by DDAT tool returns a list of conditions and for each condition a list of variables, belonging to that condition, and location where the variables have to be injected. Its output is represented in text mode. The operation Evaluate receives the list of conditions and the corresponding variable values and checks whether the conditions are evaluated appropriately. The input for the operation Apply is the BPEL process, the specific injection locations, and the specific values to inject. It calls IT of the TASSA framework by which Assign activities are inserted before the corresponding conditions in order to set the variable values. The output of the operation Apply is a transformed BPEL process, which can be executed following the desired path.

E. Test case generation tool

Test Case Generation Tool (TCGT) provides test cases for all executable paths of the BPEL process. It is responsible for storage and of the test cases and thus supports conduction of regression tests.

The relationships among TASSA tools are presented in Fig. 1.

Table I shows the correspondence of the TASSA tools to the proposed methodology. It describes the applicability of each tool to the methodology's steps.

F. Sample usage scenario of the tools

The TASSA tools described above could be used performing the following steps:

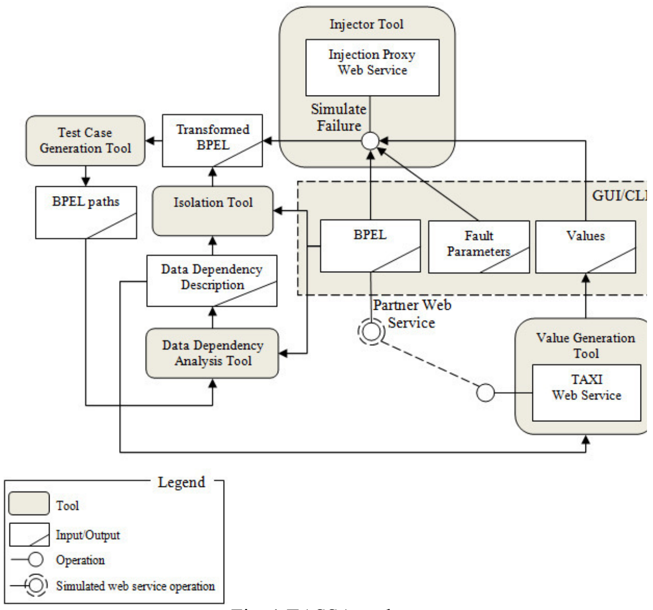


Fig. 1 TASSA tools

Step 1: Use DDAT tool to analyze the data dependencies of the BPEL process and to find the executable paths as well as their underlying variables.

Step 2: Use WS-TAXI tool to generate variable values for each path so the process execution to follow it.

Step 3: Use IT tool to replace path variables with constants, which values are generated from the WS-TAXI tool.

Step 4: If negative tests need to be performed than use FIT tool to inject faults in the BPEL process.

Step 5: If positive tests need to be performed use IT tool to remove external dependencies from partner web services of the BPEL process.

Step 6: Deploy BPEL process on an application server.

Step 7: Execute test cases.

Step 8: Compare expected with obtained results from test case execution.

IV. Validation of the Methodology

TASSA methodology is validated through usage of the implemented tools for testing of sample business processes described with BPEL.

A. Business processes used for validation

Three sample business processes are used for validation of the proposed methodology.

Order Music Process (OMP) is invokes three partner web services. Two of them, which are external, are used for e-mail validation and checking of music tracks. The third one provides offers to the customers and is implemented by the TASSA team. The process handles simple order of music tracks. It consists of four main steps. First, the customer provides his/her personal information like name and e-mail address and the music artist or album he/she is interested in. Next, a partner web service verifies the e-mail address provided by the customer. If the e-mail address is wrong, the

Table I.
Correspondence of the TASSA Tools to the Methodology

TASSA tool	Methodology step
IT	Step A.3
FIT	Step B.3
VGT	Step A.1, Step B.1, Step C.3
DDAT	Step C.1, Step C.2
TCGT	Automates the execution of all TASSA tools providing managing capabilities

business processes ends with informative message. If the e-mail is correct, the customer's order is processed. Then, the process invokes another partner web service that provides information about the available music tracks of the chosen artist or album. Finally, a third partner web service calculates the prices of the music tracks and makes an offer to the customer. If the order is less than 100 EUR the calculated price is shown to the customer and he/she can buy the music tracks of his/her favorite artist or album. If the order is greater than 100 EUR a message is sent to the customer that the company could offer him a better price for that quantity.

The Travel Reservations Process (TRP) consists of five tasks: receiving of request for reservation (receive activity), creating an airplane reservation (invoke activity), reserving a vehicle (invoke activity), reserving a hotel (invoke activity) and reply of the request (reply activity). Also the process has three conditional activities. The first conditional activity checks if there is airplane reservation for the customer. If there is not, then the process invokes the Airline Reservation web service. Otherwise, the process skips that invoke. The second conditional activity checks if there is a reserved vehicle for the customer. If there is no vehicle reservation, then the process invokes the Vehicle Reservation web service. Otherwise, the web service is skipped. The third conditional activity checks if there is a hotel reservation for the customer. If there is no reservation then the process invokes the Hotel Reservation web service. Otherwise the process continues without this web service call.

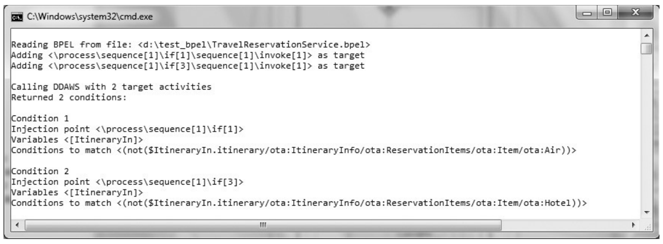
Order Data Verifier Process (ODVP) validates the clients order data, namely email, credit card number and zip code. The process consists of four web services. Email web service is used for email validation. Credit Card web service validates credit card number and type. Currency Convertor web service gets conversion rate from one currency to another currency. Zip Code web service validates a zip code and returns its USA state abbreviation, latitude and longitude in decimal degrees.

B. Validation results

The TASSA methodology was applied to all business processes presented above. In order to illustrate the usage of the TASSA tools, sample scenarios showing their functionality in the context of different business processes are chosen and presented. Complete description of the testing activities and obtained results of the methodology application can be found in [6], [7], [8], [20].

1. Data dependency analysis

Consider the TRP example. Suppose that the invocation of the Airline Reservation web service and the Hotel reservation web service need to be tested. Therefore, the process has to satisfy the first and the third conditions described in previous section. Fig. 2 presents the results form data dependency analysis performed by the DDAT [6].



```
Reading BPEL from file: <d:\test_bpel\TravelReservationService.bpel>
Adding <\process\sequence[1]\if[1]\sequence[1]\invoke[1]> as target
Adding <\process\sequence[1]\if[3]\sequence[1]\invoke[1]> as target

Calling DDAWS with 2 target activities
Returned 2 conditions:

Condition 1
Injection point <\process\sequence[1]\if[1]>
Variables <[ItineraryIn]>
Conditions to match <(not($ItineraryIn.itinerary/ota:ItineraryInfo/ota:ReservationItems/ota:Item/ota:Air))>

Condition 2
Injection point <\process\sequence[1]\if[3]>
Variables <[ItineraryIn]>
Conditions to match <(not($ItineraryIn.itinerary/ota:ItineraryInfo/ota:ReservationItems/ota:Item/ota:Hotel))>
```

Fig. 2 Data dependency analysis of the Travel Reservation Process

The returned result set consists of two conditions. Each of them is supplemented with its place in the process, and the variables that determine its outcome. This information is then used to modify the business process in order to follow preliminary specified execution path. If appropriate values (satisfying the cited conditions) of the variables are injected at the correct place (just before entering the first conditional activity), the desired web services will be invoked.

2. Web service isolation and fault injection

Suppose that ODVP needs to be isolated from its partner web services. Fig. 3 shows an invoke activity, called CardValidatorInvoke, that is responsible for invocation of web service for validation of client credit card number.

```
<invoke name="CardValidatorInvoke"
        partnerLink="CardValidatorPartner"
        operation="Validate_CreditCard"
        xmlns:tns="http://www.Softwaremaker.Net/
                  WebServices/" portType=
                  "tns:ValidatorSoap"
        inputVariable="Validate_CreditCardIn"
        outputVariable="Validate_CreditCardOut">
</invoke>
```

Fig. 3 Invoke activity before process transformation

Fig. 4 shows transformation of the above activity after execution of FIT and IT of the TASSA framework.

As can be seen from Fig. 3 and Fig. 4, the Invoke activity, named CardValidatorInvoke, is enclosed with two additional Assign activities.

The first Assign activity initializes the input parameters of the ProxyInvoke operation of FIT. The parameters are as follows:

- Serialized input arguments of card validator operation of Credit Card Validator web service;
- End point address of the Credit Card Validator web service;
- Wait interval initialized with 20;
- Error factor initialized with 0.

The second Assign activity copies deserialized result from invocation of the ProxyInvoke operation of FIT to the output variable of the Credit Card web service. In addition, CardValidatorInvoke activity invokes the ProxyInvoke operation instead actual Credit Card web service.

```
<assign name="Assign1">
  <copy><from>
 sxxf:doMarshal($Validate_CreditCardIn.parameters)
    </from><to>
      $ProxyInvokeOperationIn.operationIn/tassaP:part1
    </to></copy>
  <copy><from>
      'http://www.softwaremaker.net/webservices/
      swm/validator/validator.asmx?WSDL'
    </from><to>
      $ProxyInvokeOperationIn.operationIn/
      tassaP:endpoint
    </to></copy>
  <copy><from>20</from><to>
      $ProxyInvokeOperationIn.operationIn/tassaP:wait
    </to></copy>
  <copy><from>0</from><to>
      $ProxyInvokeOperationIn.operationIn/
      tassaP:errorsFactor
    </to></copy>
</assign>
<invoke xmlns:tns="http://www.rila.com/tassa/ProxyIn-
voke" inputVariable="ProxyInvokeOperationIn"
name="ZipCodeInvoke" operation="ProxyInvokeOperation"
outputVariable="ProxyInvokeOperationOut"
partnerLink="PartnerLink1" portType="tns:ProxyInvoke-
PortType"/>
<assign name="Assign2">
  <copy>
    <from>
  sxxf:doUnMarshal($ProxyInvokeOperationOut.part2)
    </from>
    <to part="parameters"
      variable="Validate_CreditCardOut"/>
  </copy>
</assign>
```

Fig. 4 Invoke activity after process transformation

The results form transformation of the process and execution of the generated test cases are presented in [7].

3. Negative test case generation

The last example shows test case generation for OMP when FIT is applied.

In order to illustrate the faults that can be simulated with the FIT, four test cases are defined as follows:

- Test Case 1: Message delay;
- Test Case 2: Interruption;
- Test Case 3: Noise in the message structure;
- Test Case 4: Noise in the message data.

To prove the fault injection against normal behavior of the process first additional test case should be observed:

- Test Case 0: No fault injection (normal behavior)

The generated test cases differ in the following characteristics:

- Failure parameters – describe the faults that FIT simulates. The possible failure parameters are presented in Table II.
- Activity – the activity that will be injected with faults;
- Input data – the test data put at the input of the business processes sample.

TABLE III.
FAILURE PARAMETERS

Parameter	Description
Errors Factor	Integer value defining the kind of error that will be injected (1-100 – insert 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, usually used with Data Pool Location and TAXI Web Service Endpoint Address ; -2 –interrupt the message;)
Wait Interval	an integer value that instructs the IWS how many seconds to delay the message
End Point Address	the end point address of the partner web service
Data Pool Location	the location address of predefined values that will be generated by TAXI Web Service in case of Error Factor -1
TAXI Web Service Endpoint Address	the end point address of the TAXI Web Service (TAXI-WS) that will be used to generate values in case of Error Factor -1

Each test case is validated against particular pass criteria that define the expected output of the case. The broad description of the expected result per test case is as follows:

- Test Case 0: No fault injection – the expected output without fault injection is a meaningful, well formed message that is executed for a given time interval (t).
- Test Case 1: Message delay – the expected output with a message delay is a meaningful, well formed message that is executed in time interval t + T, where T is the delay given as a failure parameter.
- Test Case 2: Interruption – the expected output when an interruption happens is an error message and the time interval is almost the same as the time interval t.
- Test Case 3: Noise in the message structure – the expected output is not a meaningful, well formed message but an error message, because of wrong structure (in few tests when the probability of appearance of corrupted data is low, the test may fall in Test Case 4)
- Test Case 4: Noise in the message data – the expected output is a well formed message that depending on the injected data will derive unexpected workflow or data values. Hence the output in this case will be similar to that in Test Case 0 with some diversion.

Following the above definition, a set of test cases was executed. The results form execution of the test cases are presented in [8].

V. RELATED WORK

This section presents an overview of the existing approaches for testing of BPEL processes. It also describes the current testing methodologies for service-based applications (SBAs) describing their key features.

A. BPEL testing approaches

The current research on testing business processes is mainly based on formal or semi-formal approaches, most of which generate abstract test cases that cannot be executed automatically.

A large number of approaches for validation of BPEL processes are based on transformation of the process under test to an intermediate model for which formal validations are well known. Transformations based on popular mathematical formalisms such as Petri Nets [9], [10], [11], process algebra [12] and state machines [13], [14], [15] are proposed. They are mainly used for static BPEL analysis. In those approaches data dependencies are not considered and produced test specifications are abstract. Therefore, an automated support for test generation and execution is not provided.

Another group of approaches rely on transformations based on Control Flow Graph (CFG) [16], [17], UML [18] and XML [19]. The framework proposed in [16] introduces a number of strategies for generation of test cases from an intermediate model which is an extension of a CFG. The strategies include full coverage, branch coverage as well as user customized test generation. The framework specified in [17] implements only basic path testing strategy to generate test specifications from a CFG. Data dependencies are derived by the BPEL process automatically but particular values should be specified by the testers. In [18] a test framework which uses UML activity diagrams to generate test specifications is proposed. Test specifications are mapped to a TTCN-3 executable test format and can be run only by tools supporting TTCN-3 format. The framework presented in [19] uses a specialized BPEL-level testing language to describe interactions with a BPEL process to be carried out in a test case. The generation of test cases is not fully automated and developers have to manually prepare a large amount of coherent XML data and XPath expression to compose a test case.

B. SOA testing methodologies

Despite the presence of various approaches and tools for testing of SBAs, the number of methodologies describing the complete process of testing is still limited.

If testing teams try to follow a classical methodology for testing, they soon will realize that it cannot be done. This is due to the fact that SOA has unique architecture ecology and own set of protocols [1]. Therefore, SOA testing tools should be able to test components without user interface and provide environment, communicate with service brokers as well as interpret message sent through Enterprise Service Bus (ESB).

A strategy for SOA testing is presented in [2]. It describes the key notions of SOA testing focusing on SOA test plan. The creation of SOA test plan starts with domain understanding and definition of testing approach. Three basic groups of testing approaches are identified, namely top-down, bottom-up and system. Since service performance is a key for success of SOA, a special attention is given to the performance testing. The proposed strategy requires testing on three main levels: Information level, Services level and Process level. All testing issues are summarized in a step-by-step guide, which consists of several steps. The first two steps require definition of testing domain and architectural objectives. Next, design review and test planning should be performed. On the fourth step, a functional testing approach is created. Further, functional performance and SLAs requirements need to be described. The next four steps require definition of approaches for data layer, service layer, policy layer and process layer testing. Further on, the strategy continues with service simulation and creation of core scenarios and user defined compliance rules. Finally, SOA testing suit should be selected, on which tests will be executed. Four types of testing are considered: Unit testing, Functional testing, Regression testing, and Compliance and validation testing. The strategy includes also looping back to design and development as well as design of approaches for design-time and run-time diagnostics.

The SOA testing methodology presented in [3] defines fours levels of testing. Service level testing requires testing of services in isolation through validation of the request-response messages according to the requirements. Process level testing includes validation of all possible process scenarios enabled by the services. The next level requires end-to-end testing, where user applications are validated according to the functional and nonfunctional requirements. The regression testing is performed on the last level in order to ensure the stability and availability of the system under test across SOA lifecycle. On the first two levels functional, security, performance and governance aspects are tested. The proposed methodology could follow bottom-up as well as top-down approach.

In [4] the SOA testing is categorized in the following phases. The first phase requires governance testing. It aims to determine if SOA policies are enforced. The second phase is performed on a service-component level. It checks if the basic functionality of the functions and components complies the specification. On the next phase a service level testing is conducted. Its goal is to ensure that the services meet project requirements as well as business and operational requirements of other processes, which use them. The integration test phase is focused on the interfaces of the services. It checks the service behavior and data exchange between services. The process level testing addresses the service orchestration. System level test phase determine if SOA technical solution provides the specified business requirements and meets the user acceptance criteria. The final test phase requires security testing. The proposed methodology also defines the types of testing that is performed on each phase. For example, the service-component test phase requires functional, performance, interoperability, backward compatibility, compliance and security testing. The interoperability and backward compatibility are missing in the service level testing.

HexSOA Test Model provides a methodology to implement and adapt 16 best practices obtained from previous SOA testing assignments of the Krosstech Solutions Company [5]. It defines the types of testing, which are performed on each phase of application development, namely functional, performance, interoperability, compliance and security. The model is based on four strategies. The unit test strategy requires unit testing to be executed for all services of the SOA system. WSDL standard, interoperability, web service security, business logic, graphical user interface (GUI) and performance benchmark should be validate on this phase. In contrast to other methodologies, the business process testing is also performed on this phase. Service emulation is proposed in case of missing or disabled services. Integration test strategy aims to validate integration layers of the SOA system and its pathways. Integration layers include web services, middle-tire services, services exposing functionality of the legacy systems, etc. This phase is more focused on the business processes rather than on the code and GUI. Functional test automation strategy aims to identify recordable test cases, create test scripts, interpret test results and report them to the development team. Performance test strategy requires performance end to end testing of the SOA system.

The presented above methodologies provide overview of the testing activities performed on different layers of the service-oriented application under test, which are called levels or phases. They present SOA testing from a high level perspective, where details about the testing activities performed on each level is missing. In contrast, TASSA methodology is focused on the business process level testing providing description of the required activities in a step-by-step manner. It defines concrete testing approaches and shows not only what should be done but how it should be done as well. Since a lot of tools for functional testing of single web services exist, the proposed methodology does not consider testing activities on a web service level.

VI. Conclusion

The TASSA framework is a methodology and a set of tools for testing business conforming WS-BPEL standard. It complements existing development environments' native verification tools and can be used jointly to achieve end-to-end design-time testing of service based applications. The TASSA framework is validated through testing of sample business processes. The obtained results show the effectiveness of its capabilities for functional testing as well as for robustness testing via injection of invalid, unexpected or random data into a business process.

The future work will be focused on extension the proposed methodology towards run-time testing of BPEL processes. The work on the approach for monitoring of business processes is started. The goal of the monitoring is not only to estimate the quality characteristics of the business process, but also to find patterns to predict possible failures.

References

[1] G. Hattangadi, "A Practical Guide To Modern SOA Testing," White paper, July, 2011.

[2] D. Linthicum and J. Murphy, "Key Strategies For SOA Testing," Mindreef, Inc., Hollis, New Hampshire, 2007.

[3] G. Hattangadi and R. Gupta, "Driving Better Business Process Scalability with Modern Software Quality," white paper, July, 2010.

[4] "SOA Test Methodology," white paper, Torry Harris Business Solutions, July, 2007.

[5] S. Shriram, "The HexSOA Test Model," white paper, Krosstech Solutions, August, 2009.

[6] I. Spassov, D. Petrova, V. Pavlov, S. Ilieva, “DDAT: Data Dependency Analysis Tool for Web Service Business Processes,” Second International Workshop on “Software Quality SQ” within The International Conference on Computational Science and Applications (ICCSA), Santander, Spain, B. Murgante et al. (Eds.): ICCSA 2011, Part V, LNCS 6786, Springer, Heidelberg, pp. 232-243.

[7] D. Petrova-Antonova, S. Ilieva, I. Manova, D. Manova, “Towards Automation Design Time Testing of Web Service Compositions,” The 5th IFIP TC2 Central and Eastern European Conference on Software Engineering Techniques (CEE-SET), Debrecen, Hungary, August, 2011.

[8] D. Manova, I. Manova, S. Ilieva, D. Petrova-Antonova, “faultInjector: A Tool for Injection of Faults in Synchronous WS-BPEL processes,” 2nd Eastern European Regional Conference on the Engineering of Computer Based Systems (ECBS-EERC), September, 2011, Bratislava, Slovakia, pp. 99-105.

[9] C. Ouyang, E. Verbeek, W. Vanderaalst, S. Breutel, M. Dumas, A. Terhofstede, "Formal Semantics and Analysis of Control Flow in WS-BPEL," Science of Computer Programming, Vol. 67, No. 2-3, July 2007, pp. 162-198.

[10] Sebastian Hinz, Karsten Schmidt, and Christian Stahl, "Transforming BPEL to Petri Nets," In Wil M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, Proceedings of the Third International Conference on Business Process Management (BPM 2005), volume 3649 of Lecture Notes in Computer Science, Nancy, France, September 2005, pp. 220-235.

[11] Chien-Hung Liu, Shu-Ling Chen, Xue-Yuan Li, "A WS-BPEL Based Structural Testing Approach for Web Service Compositions," IEEE International Symposium on Service-Oriented System Engineering, 2008, pp.135-141.

[12] H. Foster, S. Uchitel, J. Magee, J. Kramer, "LTSA-WS: a tool for model based verification of web service compositions and choreography," In Proceeding of the 28th international conference on Software Engineering (ICSE) – Research Demonstration, pp. 771–774.

[13] J. Garc´ıa-Fanjul, J. Tuya, and C. de la Riva, “Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN,” in Proc. of WS-MaTe, 2006, pp. 83-94.

[14] Yuan Yuan, Zhongjie Li, Wei Sun, "A Graph-Search Based Approach to BPEL4WS Test Generation", Proceedings of the International Conference on Software Engineering Advances, October 29-November, 2006

[15] Xiang Fu, Tevfik Bultan, Jianwen Su, "WSAT: A Tool for Formal Analysis of Web Services", the Proc. of 16th Int. Conf. on Computer Aided Verification, pp. 510-514.

[16] Z. J. Li, H. F. Tan, H. H. Liu, J. Zhu, N. M. Mitsumori, "Businessprocess-driven gray-box SOA testing", IBM Systems Journal, v.47 n.3, July 2008, pp.457-472.

[17] T. Lertphumpanya, T. Senivongse, "Basis path test suite and testing process for WS-BPEL," WSEAS Transactions on Computers, vol. 7, issue 5, 2008, pp. 483-496.

[18] Qiulu Yuan; Ji Wu; Chao Liu; Li Zhang, "A model driven approach toward business process test case generation," 10th International Symposium on Web Site Evolution (October 2008), 2008, pp. 41-44.

[19] Philip Mayer, Daniel Lübke, "Towards a BPEL unit testing framework," Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications, July 17-17, 2006, Portland, Maine, pp.33-42.

[20] D. Manova, S. Ilieva, F. Lonetti, A. Bertolino, C. Bartolini, “Towards Automated Robustness Testing of BPEL Orchestrations,” International Conference on Computer Systems and Technologies (CompSysTech), June, 2011, Vienna, Austria, pp. 659-664.