

## Testing Web Service Compositions: Approaches, Methodology and Automation

Dessislava Petrova-Antonova<sup>1,\*</sup>, Denitsa Manova<sup>2</sup>, Sylvia Ilieva<sup>1</sup>

<sup>1</sup>Software engineering, Sofia University, 1113, Bulgaria

<sup>2</sup>Rila Solutions, 1113, Bulgaria

---

### ARTICLE INFO

Article history:

Received: 05 November, 2019

Accepted: 20 December, 2019

Online: 22 January, 2020

---

Keywords:

Testing approaches

Testing methodology

Testing web service compositions

---

### ABSTRACT

Web services give a new view of the web as the biggest, widely accepted and the most straightforward distributed software platform. Their composition into applications and business processes is still a complex, non-trivial task, requiring highly rational efforts not only from the software developers, but from the quality assurance specialists. The provision of web service compositions' quality brings a lot of challenges due to variability of difficulties at infrastructure, service and choreography levels and the need of different types of testing in unknown context and environment. A consolidated quality assurance methodology that advances the fundamental understanding of testing in terms of concepts, models, techniques, standards and automation is required. This methodology needs to enable effective exploration, comparison, evaluation and selection of testing approaches, platforms and tools.

This article proposes such a methodology and reviews the current testing approaches for single and composite web services from an objective, holistic perspective. The methodology is presented as an end-to-end testing procedure, in which activities are facilitated by a set of testing approaches, techniques and best practices. A concrete solution is recommended for actual implementation of each activity either through selection among the most appropriate and effective existing approaches or development of new approaches, mainly in case of critical issues such as dependencies analysis, partner web services' isolation and injection of faults. A common framework that integrates different testing tools automates the methodology. Its applicability, completeness, level of automation, and level of novelty is evaluated through testing of real testing scenarios.

---

## 1. Introduction

Web services provide a novel paradigm for interoperability of distributed applications. They act as collaborative agents to deliver advanced and high-quality functionality to the end users. That is why the web services are used by the successful enterprises to build flexible and fast connections with their partners, and thus reducing the cost and increasing the revenue. Such business interactions are possible through implementation of Web Service Compositions (WSCs) in a standardized, secure and interoperable manner. Although the WSCs provide a lot of benefits for the developers, they bring challenges to testing as well as to overall quality assurance process. First, its implementation is often based on web services, which are developed and deployed on diverse

environments supported by different vendors. In case of legacy systems and components, the web services act as wrappers of functionality that is hard to be controlled and simulated in a testing environment. The orchestrated web services are not always available or could be undeployed by the provider, which leads to additional efforts for emulation of their behavior during testing. In addition, the emulation is required in case of payed web services to reduce the testing process's cost. Appropriate message data needs to be generated not only for the emulated web services to mimic their behavior, but for the composition as whole in order to be invoked in the testing environment. Achieving a high level of test coverage and determination of failure causes is difficult due to the missing programming code of the orchestrated services as well as the requirement for creating test cases based only on their public interfaces. Such interfaces are usually defined using the Web Service Description Language (WSDL). When a dynamic binding

---

\*Dessislava Petrova-Antonova, 1113 Sofia, Bulgaria, 125 Tsarigradsko shose Blvd. Blok 2, +359887572094 & d.petrova@fmi.uni-sofia.bg

of web services is implemented, the emulation process becomes more complex, since advanced stubs (mockups) should be developed.

A lot of testing approaches and automated tools have been proposed to meet the above challenges. Unfortunately, the most of them cover only single testing activities such as analysis of testing paths, generation of test cases, web service mocking, injection of faults, etc. However, it is important to merge all testing activities in a general testing procedure supported by end-to-end methodology and thus to facilitate the whole testing process. This article is an extension of work originally presented in International Conference on the Quality of Information and Communications Technology (QUATIC'18) [1], which proposes such a methodology, called Testing As Service Software Architecture (TASSA). The methodology addresses the following major problems of WSC testing:

- Difficulties in emulation of web services deployed on heterogeneous platform and controlled by external providers;
- Missing or temporally unavailable web services and inability to detect the causes of failures;
- Lack of automation to test unanticipated behavior of web services and test case generation to reach a high level of test coverage.

Therefore, the benefit of the proposed methodology are as follows:

- Provision of end-to-end testing methodology for WSC;
- Recommendation of concrete approaches for actual implementation of each activity of the methodology;

- Development of completely new approaches to cover critical issues such as analysis of data dependencies, isolation of partner web services and injection of faults;
- Full automation in a single framework for testing WSCs, described with Business process execution language (WS-BPEL) [2].

The next two sections present the TASSA methodology and the testing approaches appropriate for its implementation. Section 4 provides a view of TASSA methodology's validation and Section 5 summarizes the validation results. Section 6 concludes the paper.

## 2. TASSA methodology

TASSA methodology supports functional, performance and security testing of WSCs and provide means for validation of their behavior if implementation changes are in place. It relies on a small number of artefacts, such as a BPEL file, input data, expected output data and test assertions. It consists of seven main activities, presented with a workflow diagram in Figure 1.

### 2.1. Prerequisites

W3C defines a set of standards for WSCs that are used for standard compliance validation of the WSCs [3]. If some inconsistencies are detected, a notification to perform the appropriate corections are sent to the tester. The isolation of a partner web service can can follow two approaches: (1) invocation of mockup service that mimic the behavior of the partner web service or (2) generation of expected response that is expected from the partner web service. TASSA methodology adopts the second approach based on change of the communication channel between the WSC and its partner web service. This is realized using a so called "mediator service" allows fo injection of different types of faults in the communication channel.

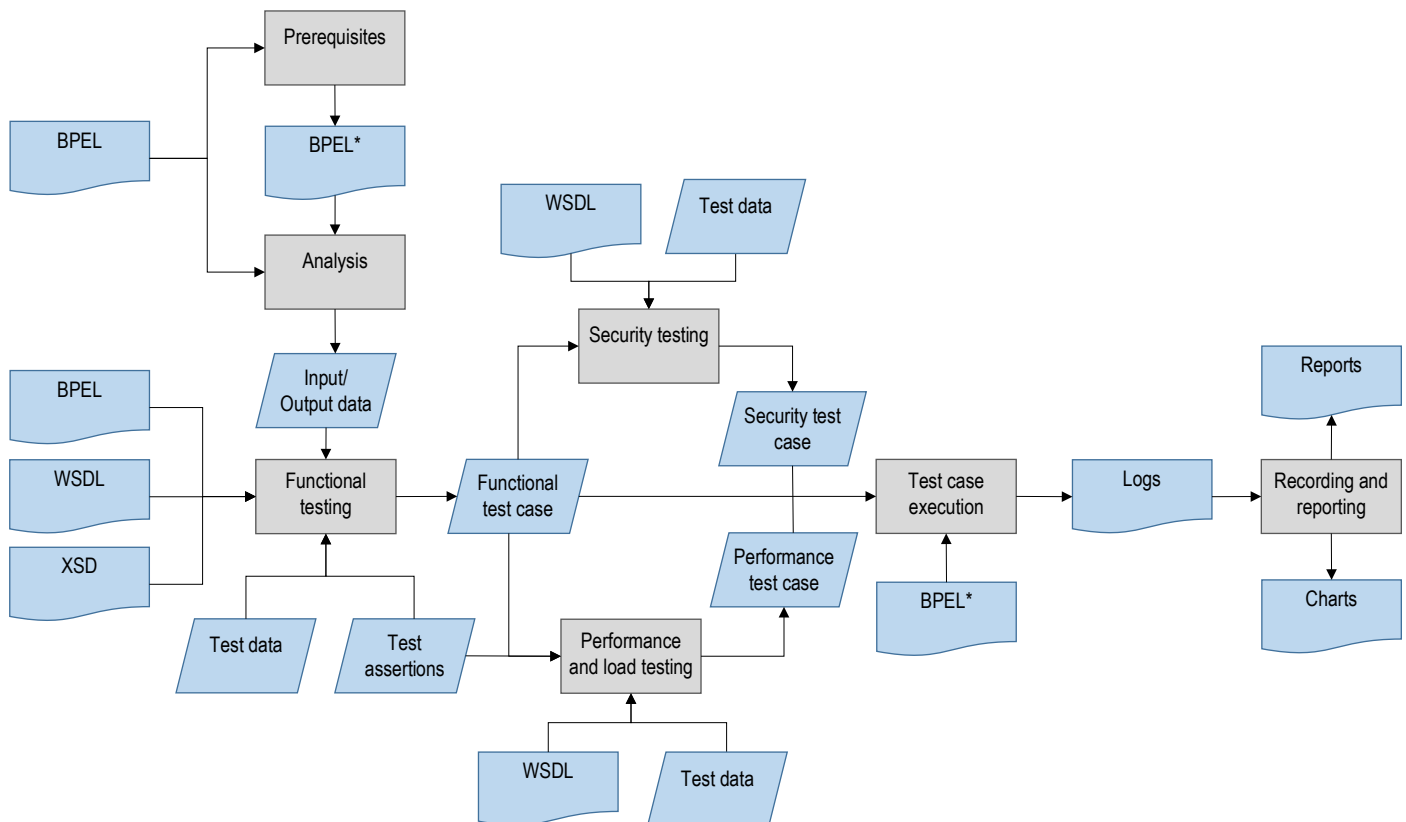


Figure 1: Methodology workflow

## 2.2. Analysis

In order to obtain all test paths, the WSC is usually represented using a specific formal model. TASSA methodology transforms the WSC in a BPEL Flow Graph (BFG) similarly to the approaches in [4] and [5]. First, all paths in the BFG are identified, next the unfeasible ones are filtered and the finally the feasible paths are used for generation of test cases.

## 2.3. Functional testing

The WSDL description of the WSC is the primary source for functional test case generation. The main components of a given test case are the request with input data and the expected response with output data. Similar test cases are a common practice for testing single web services. A modified approach that support tracing of the executed activities in the WSC is applied by the TASSA methodology.

The generation of test data is based on a widely adopted approach based on XML Schema Definition (XSD) of the WSC. It is further developed to support parametrization of test cases using variables, functions as well as connectivity with external data pools allowing for data-driven testing. The expected results from the execution of the WSC are a base for definition of test assertions. The execution order of the activities in the WSC is an important assertion, when the behavior of multiple web services is orchestrated. Additional assertions relevant to both single and composite web services are defined, based on a single value, a value type, a regular expression, an HTTP status code, a file type and size, etc.

## 2.4. Security testing

The request to the WSC is enriched with appropriate test artefacts for the purposes of security testing. The response from the WSC is checked whether it matches the security requirements, such as delivery of partial message, unauthorized access, unavailable partner web service, etc. At the message level, the security testing includes sending of "secure" request and checking the response sent by the web service. There are a lot of security mechanisms applicable to WSCs. The TASSA methodology proposes usage of WS-Security standard and QoS policies, which are based on enrichment of exchanged messages with a specific information based on WS-RM, WS-Addressing or MOTM. At transport level, the testing for security is related to testing of the authentication, including sending certificates like Kerberos, SSL, etc.

The TASSA methodology follows a new approach for the robustness testing. It transforms the WSC under test allowing for simulation of faults during its execution. The supported faults are an unavailable partner web service, a delay of the response from a partner web service, wrong structure or semantic of the response from a partner web service. The test assertions for security testing are defined in a similar way of functional testing.

## 2.5. Performance and load testing

The performance and load testing validate the non-functional characteristics of the WSC. Since, it requires a set of requests to be send in order to simulate parallel executions, a definition of virtual users is performed for single functional test case or a group of them

for a given time interval. The threads are commonly used in the most cases. They are scheduled to define the way, on which the virtual users are managed. The schedule contains information about the time for starting and stopping of requests, execution order of requests and time interval between them, number of executions, etc.

The test assertions for performance testing are based on those for functional testing. Test assertions for time intervals, resource usage, file sizes and others characteristics are also defined. They provide insight for the system and network load, the hardware and software resources, etc. Additionally, a special assertion for checking the compliance with the Service Level Agreements (SLAs) is created.

## 2.6. Test case execution

Prior execution of functional test cases, the BPEL file of the WSC should be deployed on the application server and an instance of the WSC should be created. The request defined in the functional test case is sent to the WSC and its behavior is tested in a similar way to a single web service. The security testing requires to configure the security settings of the application server, where the BPEL file is deployed. The performance or load testing needs definition and management of threads according to the parameters of in the test cases. When a huge load needs to be simulated or hardware resources of the application server are limited, the execution of test cases is distributed among different machines.

## 2.7. Recording and reporting

The information obtained during the execution of the test cases is recorded in a log file. In order to allow for easily processing, the valuable outputs are collected in a test data storage. Apart from the directly obtained test results, additional metrics are recorded, which are related to the WSC itself, the application server, communication channels, the network and so on. These metrics are simple, but can be further aggregated for computing of more complex ones. In order to check whether the test is failed or passed, the expected output and actual one after test case execution are compared based on the test assertions. When the test is failed, the reason for the failure is collected for defect allocation as well as for a subsequent regression testing. The test reporting uses the data from the log files and the corresponding test data storage. The XML is the most commonly used data format for the test reports.

# 3. Testing Approaches

This section presents testing approaches that are appropriate for implementation of TASSA methodology. For each methodology's task several approaches are identified and the most appropriate ones are recommended. For particular tasks new approaches are developed and reasoning about their applicability is given.

## 3.1. Standard compliance checking

The preparation of WSC for testing includes three main activities, described in previews section, namely checking for compliance with standards, isolation of partner web services and injection of faults.

The compliance with the standards can be validated by the most of Integrated Development Environments (IDEs) such as

Eclipse BPEL Validator, Oracle ILINT and NetBeans XML Validate. It can be performed following three different techniques depending on the validation rules:

- Rules described with a programming code (e.g. code written with Java);
- Rules based on XML or OCL constraints;
- Rules defined with a model of the WSC description (e.g. UML model).

The proposed methodology relies on the Eclipse BPEL Validator. It processes the BPEL file as a DOM and for each node (assign, copy, from, to, etc.) creates rules for checking the constraints prescribed by the corresponding standard. The rules are written in Java and can be changed by the tester.

### *3.2. Isolation of partner web services*

The replacement of partner web service with a stub (mockup) service is a technique widely used to remove external dependencies from the WSC [6,7]. An alternative approach is to generate a message that substitutes the response expected by the WSC from a partner web service [8,9,10]. The second approach requires the real communication with the partner web services to be recorded and appropriate interfaces and data to be manually defined.

For the purpose of TASSA methodology, a new approach is elaborated. It replaces the BPEL activities related to external dependencies (e.g. Invoke activities) with ones that are internal to the WSC and provide similar output to the original activities (e.g. Assign activities). Since the output of the simulated activities are known, the execution of the WSC can be controlled to follow a particular path. For this purpose, proper values for the variables in conditional branches should be defined. Identification of the sequence of branch conditions and the values for conditional variables is a heavily task but it is automated by the TASSA isolation tool.

### *3.3. Fault injection*

The robustness of the WSC can be tested relying on methods for negative testing by causing faults in the WSC itself or in its communication with the partner web services. A classification of the faults that are typical for service-oriented systems can be found in [11], while the specific faults for WSC are described in [12]. In [13] authors propose an approach that injects faults in the partner web service and then perform coverage testing of the code responsible for recovery from failures (e.g. exception handlers). In [14] a fault injection technique is applied by exploring the behavior of the WSC regarding the injected faults and thus to assess the its fault tolerance capabilities.

The fault injection is the second new approach proposed by the TASSA methodology. It supports injection of delays, errors and other malfunctions in the message exchange between WSC and its partner web services. The BPEL file of the WSC is modified by replacing the call to a partner web service with a call to a proxy service. In case of faults, the behavior of the third-party services is out of the TASSA scope. Therefore, faults in the outgoing calls are not injected. The proposed approach requires the original partner web service to be called and then intervention in its response to be performed. For this reason, the proxy service calls the original partner web service and then using the information for the fault

simulated performs the required action. When data errors need to be injected, the two options are available – to change the values of data fields, while keeping the structure of the message or to insert random errors in the message data, which could break the XML structure of the response. For the first case, a tool for random generation of data according to an XML schema is required.

### *3.4. Dependency analysis*

The analysis of WSC is focused on states, transitions and related usage of BPEL activities that are part of complex interactions. The software systems support two kinds of interactions as follows:

- Data flow interactions, where the definition and values of the next data items depend on the definition and values of the previous ones.
- Control flow interactions along the execution path, in which the next executions depend on the previous ones;

The testing of the above interactions is covered by the data flow testing and control flow testing, respectively. Such techniques are applicable mainly to white-box testing. In the context of the service-oriented systems, there are additional interactions due to communication with external services, for which only the types of input and output data are known.

An overview of types of interactions, related dependencies and approaches for analysis is can be found in [15]. The current approaches for dependency analysis apply variety formal models. The most popular ones transform the WSC in a control flow graph [16,17], Petri net [18,19], a state graph [20] or an UML model [21,22].

For the purpose of TASSA methodology, a new approach for dependency analysis is developed. As was mentioned before, it transforms the WSC in a BFG in order to find all executable paths. The dependency analysis includes identification of the conditional statements together with their conditional variables and constants, which values guide the execution of the WSC on a certain path.

### *3.5. Test data generation*

The test data generation is a complex task, since the programming code of the WSC itself and its partner web services is often unavailable. That is why the approaches for test data generation are “black box” based, meaning that they rely only on the scope and type of the input and output variables.

Since the communication in WSCs is performed through exchange of XML messages, a large group of test data generation techniques use WSDL descriptions of the web services in combination with their XSDs [23,24,25,26,27]. The XSDs determine the constraints over the data types that are useful for generation of both simple and complex test data.

The approach proposed for TASSA methodology is similar to those presented in [28,29,30,31], since it also processes XSDs. It produces XML instances from a given XML Schema available in the WSDL file or in the BPEL file. Thus, XML messages needed for communication of the WSC with its partner web services are generated. The proposed approach can be applied to functional and robustness testing of WSCs due to support of both correct and incorrect XML instances’ generation. The implemented algorithm supports generation of empty XML documents – XML instances, which structure follows a given XML Schema, but does not



contain concrete values for the XML elements. In addition, generation of XML instances populated with random valid and invalid data depending on the testing goal is provided. Finally, a manual specification of values for the XML elements or their loading from external file are supported.

### *3.6. Test assertions definition*

The service-oriented systems require definition of test assertions for a large group of test artefacts such as file content, file properties, message content (both header and body), message properties, expressions (e.g. XPath), variable values, variable properties, time properties and sequences of activities' execution.

Durand and Green propose a test script model allowing for expression of predicates crossing over diverse inputs and to handle a richer spectrum of outcomes [32]. The test assertions are well-structured objects defined with identifier, source (normative conformance requirement), target (instance of a specific artifact type), pre-condition that should be fulfilled, predicate (logical expression over the target) and prescription level (level of imperativeness of the source requirement). The major benefit of such approach is that the target can be tested within its context. The test assertions are defined using XLST and XPath and can be applied to functional testing and non-functional testing.

The power of XPath functions and expressions is leverage by Schematron – a simple pattern and rule language suitable for document testing [33]. Since the handling of the namespace is difficult, when rules are written according to an XML schema, and when there is a namespace prefix in a value of an attribute or element, the language is primarily intended for XML instances. XMLUnit is a tool offering both Java and .NET interfaces, which also supports validation according to a given XML schema, assertion of the values of XPath queries or comparison of XML documents with the expected outcomes [34]. The approach behind the tool relies on JDOM to traverse the XML tree.

TASSA methodology adopts the approach proposed by Durand and Green. In addition to it, the XMLUnit is also utilized due to its high level of automation.

### *3.7. Functional test case generation*

The approaches for generation of functional test cases can be divided in two groups depending on the type of testing technique – white box or black box. The black box techniques handle the WSC as a single service with known operations and message types defined in a WSDL document. On the other hand, the white box techniques explore the actual BPEL file to find information about activities, control flow and data flow. According to the formal model used for transformation of WSC to find executable paths and thus generate test cases, the approaches falls in three main groups: approaches based on CFG [16], approaches using Petri nets [18] and approaches adopting model checking technique such as Web Service Automata (WSA) [35], Stream X-machine (SXM) [36], UML 2.0 activity diagram [37], PROMELA [38], etc.

The solution for test case generation of TASSA methodology follows mixed approach – a combination of black box and white box techniques. It supports the following tasks: (1) determination of web service operations and BPEL variables; (2) generation of templates of SOAP request; (3) test assertions' definition at HTTP level, SOAP level and BPEL variable level; (4) execution of test cases by exchange of SOAP messages; (5) collection of test results for reporting and follow-up actions. The test cases are described

and stored in an XML format, where the root element has two attributes – “name”, corresponding to the test case name, and “template”, indicating whether a data-driven testing can be performed. The other elements are a narrative description of the test case, a WSDL operation under test, an input data, a data source for the purpose of data-driven testing, and test assertions. The provided types of assertions verify the status code of the HTTP response and response time, validate the SOAP message in case of success or failure and can apply XPath expressions to a BPEL variable or to a SOAP body.

The test case provides information for the web service address, and the message parts like HTTP headers, authentication mechanisms if they are required, and data that is carried in the body of the SOAP request. The test result consists of HTTP headers, data that is carried in the body of the SOAP response, the BPEL variables' values and the execution time. In case of data-driven testing, parametrized test cases are generated, in which the actual data in the SOAP request body and the expected data in the SOAP response body are substituted with variables.

### *3.8. Performance testing*

Since the web services are inherently concurrent, the concurrent issues continue to receive a huge interest. There are two primary solutions for implementation of concurrency models, namely event-based and thread-based. In thread-based applications each request is resolved in synchronous mode. Thus, each request is processed in a single thread. When the response is complete, the thread returns back to thread pool. If an external service is called to resolve the request, the thread has to wait for the response. In event-based applications each request is resolved in an asynchronous mode. The thread returns back to thread pool before the response is completed and it is ready to serve some other request. There is a single thread (or a small number of threads) that routes and manages all requests. If an external service is called to resolve the request, the process continue to execute without blocking and receives a response through a callback mechanism. The BPEL engines work following one of the two models. IBM WebSphere Process Server, ActiveBPEL Engine, Oracle BPEL Process Manager are thread-based, while BPWS4J and BPEL-Mora are event-based. A BPEL engine prototype with a high performance, proposed in [39], adopts an event-driven architecture and join patterns delivered by the Microsoft Concurrent Coordination Runtime (CCR). Apache ODE executes long-running business processes, described with BPEL. ODE's Java Concurrent Objects (JaCOB) framework ensures the runtime implementation of BPEL constructs at the instance level [40]. JaCOB supports a concurrency at application-level without relying on threads.

The type of concurrency model directly affects the performance of WSC. In addition to the Response time, other non-functional characteristics of web services related to their performance are Throughput, Availability, Accessibility and Successful rate. In the context of WSC, the performance depends on the orchestration of the partner web services, which can follow five patterns: sequence parallel, conditional, cyclic or discriminative execution. For each execution pattern, the quality factor of each partner service is calculated and then an aggregated factor for the whole WSC is delivered. The Response time, Availability, Reliability, Bandwidth and Cost are non-functional characteristics used for assessment of WSCs in [41]. A drawback of the proposed approach is that it considers only sequence

execution of partner web services. The approach presented in [42] covers all five execution patterns, but takes into account only three are non-functional characteristics (Response time, Reliability and Cost). In contrast to the approaches that apply aggregation method for quality assessment, there is a more realistic method, which accounts for the uncertainty of partner web services and estimate the quality of a WSC's probabilistically.

TASSA methodology proposes usage of combination of approaches. It is validated in the context of Apache ODE BPEL Engine. The generation of the virtual users is performed by external tool, which handles the WSC as a single service applying black box testing. The same number of requests following the same schedule are used for testing of WSC and its partner web services.

### 3.9. Security testing

The security testing of WSCs brings challenges due to their distributed nature and requires application of more than a single approach. Kabbani and Tilley consider the security of WSCs from three points of view: known threats and weaknesses, security mechanisms for web services and privacy, integrity and accessibility [43]. The Open Web Application Security Project (OWASP) works on improving the software security. It describes a set of security vulnerabilities, including those that affect the web services. It proposes tools and best practices for security assurance [44]. The OASIS consortium describes security quality factors such as Encryption, Authorization, Authentication, Non-Repudiation, Integrity, Availability, Privacy and Audit [45]. The WS-standards (WS-Security Policy, WS-Addressing, WS-SOAP Message Security, WS-Trust) prescribe mechanisms for secure communication between single web services, but they are applicable to WSCs too.

Several research works propose development of models for data access of single services [46,47,48]. The security assessment of WSCs is based on such models. An alternative approach is to use a web service mediator, called broker, which composes the web services against the specified security constraints. An optimal WSC is selected according the QoS properties [49]. The broker uses a repository to store security and quality properties of web services and updates these properties after each execution of WSC. The security certification of web services is a technique applicable to selection, discovery, and composition processes. It includes certificate issuing and management, certification-aware service discovery, certificate validation, and service consumption [50]. Biskup et. al propose a framework supporting a decentralized execution of WSCs, which guarantee the correctness and the security of the execution [51]. The approach is based on a container – an encrypted and authenticated data structure that is passed among the composed web services. An XML-based script language for definition of security policies of WSCs is presented in [52]. It is integrated to a platform for run-time monitoring and security analysis. Data sharing agreements are another solution for secure collaboration between parties [53]. They define the data sharing policies for authorizations, obligations and prohibitions. The policies indicate the authorized, obliged or denied actions together with their related data and subject.

TASSA methodology relies on validation according to the security standards, testing through fault injection, described in Section 3.3 and checking of the implemented security mechanisms. The quality factors of security in [45] are defined for single web services. The security tests check whether particular mechanisms and standards are applied or not. They can be

performed on a transport level or on a message level. Since TASSA methodology are focused on WSCs, it proposes testing of the data exchange security, as it is prescribed in [53].

### 3.10. Recording and documentation of test results

The instrumentation is a popular method for monitoring and analysis of the execution of WSCs is [54,55,56]. It adds a specific code in the BPEL file, which collects execution information without changing the normal execution of the WSC. The collected information is sent to external auditing web service for processing. The approach is technology independent and is suitable for functional testing, since it allows monitoring of data flow and control flow. It can be applied to security testing, but it is not appropriate for performance testing due to invocation of external web service, which can slow down the performance.

The BPEL engine itself provides a detail information for the execution state of the WSC. For example, the ActiveBPEL Engine [57], Oracle BPEL Process Manager (Glassfish) [58], IBM WebSphere Process Server [59], Apache ODE [40] collect information about the Response time, the variables and activities as well as the invoked partner web services and the respective message exchange. Since the BPEL engines control the execution of WSC, the approaches and tools relying on their capabilities are technology dependent, but applicable to both functional and non-functional testing. SALMon is a service-oriented system for monitoring services in order to detect violations in service level agreements (SLAs) [60]. It is technology independent, including three types of components, namely Monitors, Analyzer and Decision maker. The Monitors consist of measuring instruments, the Analyzer checks the SLA rules, while the Decision maker performs corrections to satisfy SLA rules. An architecture for run-time monitoring of WSCs, described with BPEL, is proposed in [61]. The business logic of the composition and the monitoring functionality are clearly separated through implementation of two types of monitors – instance monitors and class monitors. The instance monitors are responsible for execution of a single instance of the composition, while the class monitors report aggregated information about all executed instances. Both monitors are specified through a specific language and are generated as Java programs, which can be deployed in the run-time environment of the monitor engine.

TASSA methodology adopts the approach relying on the BPEL engine for monitoring of WSC and subsequent reporting of the collected information. Such approach is suitable for all types of testing and provides information about assertions, variables, messages, control flow and communication with the partner web services. Its advantage is the lack of intervention in the WSC and early reporting of the test results.

## 4. Validation of TASSA methodology

The validation of the TASSA methodology is performed through its application for testing of a sample business process, called *Check Payment*. The testing is performed using NetBeans integrated development environment.

### 4.1. Functional testing

The definition of functional test cases includes the following steps:

- 1) Selection of BPEL file, containing description of the *Check Payment* business process;

- 2) Selection of test path from the data dependency tree, generated from the BPEL file;
- 3) Identification of variables and constants, from which the execution of the selected path depend on, using the data dependency analysis tool. The tool returns description of dependencies, shown in Listing 1;

Listing 1: Description of dependencies

```
This branch is ELSE
This branch is IF
This branch is IF Condition 0:
String representation:
(count($OrderPartnerOperationIn.Order_Input/ns0:state) = 0
or count($OrderPartnerOperationIn.Order_Input/ns0:city) = 0)
Depending on: OrderPartnerOperationIn
Located at:
\process\sequence[1]\if[1]\sequence[1]\if[1]\sequence[1]\if[1]
Condition 1: String representation: (true() =
$ValidateEMailOut.parameters/ns6:ValidateEMailResult)
Depending on: ValidateEMailOut
Located at: \process\sequence[1]\if[1]\sequence[1]\if[1]
Condition 2:
String representation:
($Validate_CreditCardOut.parameters/ns1:Validate_CreditCardResult = 1)
Depending on: Validate_CreditCardOut
Located at: \process\sequence[1]\if[1]
```

Listing 2: SOAP request

```
<SOAPEnv:Envelope xsi:schemaLocation="..."
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPEnv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="..." xmlns:ord1="http://xml.netbeans.org/schema/OrderInfo">
<SOAPEnv:Body>
<ord:OrderPartnerOperation>
<Order_Input>
<ord1:firstname?string?/></ord1:firstname>
<ord1:lastname?string?/></ord1:lastname>
<ord1:email?string?/></ord1:email>
<ord1:country?string?/></ord1:country>
<!--Optional:-->
<ord1:state?string?/></ord1:state>
<!--Optional:-->
<ord1:city?string?/></ord1:city>
<ord1:postalCode?string?/></ord1:postalCode>
<!--Optional:-->
<ord1:address?string?/></ord1:address>
<!--Optional:-->
<ord1:phone?string?/></ord1:phone>
<ord1:creditCardNumber?string?/></ord1:creditCardNumber>
<ord1:total?1.051732E7?/></ord1:total>
<ord1:currencyCode?string?/></ord1:currencyCode>
<!--Optional:-->
<ord1:errorCode?string?/></ord1:errorCode>
</Order_Input>
</ord:OrderPartnerOperation>
</SOAPEnv:Body>
</SOAPEnv:Envelope>
```

Listing 3: Input test data

```
<SOAPEnv:Envelope xsi:schemaLocation="..."
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPEnv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ord="..." xmlns:ord1="http://xml.netbeans.org/schema/OrderInfo">
<SOAPEnv:Body>
<ord:OrderPartnerOperation>
<Order_Input>
<ord1:firstname>John</ord1:firstname>
<ord1:lastname>Doe</ord1:lastname>
<ord1:email>john@gmail.com</ord1:email>
<ord1:country>USA</ord1:country>
<ord1:postalCode>77590</ord1:postalCode>
<!--Optional:-->
<ord1:address>St. John 114</ord1:address>
<!--Optional:-->
<ord1:phone>00359111222</ord1:phone>
<ord1:creditCardNumber>41111111</ord1:creditCardNumber>
<ord1:total>40</ord1:total>
<ord1:currencyCode>USD</ord1:currencyCode>
</Order_Input>
</ord:OrderPartnerOperation>
</SOAPEnv:Body>
</SOAPEnv:Envelope>
```

- 4) Generation of abstract test case for testing of business process's operation *OrderPartnerOperation*. The SOAP request to the partner web service is shown in Listing 2;

- 5) Creation of executable test cases from the abstract one by manual or automated specification of values for the input variables, considering the identified constraints on step 3. The expected results and assertions are defined using the test case generation tool. In order to execute the path that contains the activity *CityAssign*, the credit card number and the client email should be valid, while the city and the state may be omitted. The input data that satisfy these conditions as well as standard data types in the XSD schema of the business process are presented in Listing 3;

The following assertions are defined:

- HTTP status code – checks the HTTP status code of the response. The expected value is 200;
  - XPath equals – the BPEL variable *ValidateEMailOut* should contain XPath expression *message/part/ValidateEMailResponse/ValidateEMailRe* with value *True*.
  - XPath equals – the BPEL variable *Validate\_CreditCardOut* should contain XPath expression *message/part/Validate\_CreditCardResponse/Validate\_CreditCardResult* with value *1*.
  - Not XPath exists – the BPEL variable *OrderPartnerOperationIn* should not contain XPath expression *message/part/city*.
  - Contains – the response is expected to contain the regular expression *<ord1:city>.\*</ord1:city>*;
  - Response time – the maximum value of the Response time is expected to be 5 000 ms.
- 6) The test cases are executed and the results are recorded and analyzed.

Table 1: Output from functional testing

No	Assertion	Result
1	HTTP status code	200
2	XPath equals on <i>ValidateEMailOut</i> variable	True
3	XPath equals on <i>Validate_CreditCardOut</i> variable	1
4	Not XPath exists on <i>OrderPartnerOperationIn</i> variable	XPath not found
5	Contains (case sensitive: "false", is regex: "true")	Value found
6	Response time	2562 ms

#### 4.2. Isolation of partner web service

Since the partner web service *addresslookup* allows limited invocations over time, it is isolated from the business process by performing the following steps:

- 1) Selection of BPEL file, containing description of the *Check Payment* business process.
- 2) Selection of *addresslookup* web service for isolation;
- 3) Formal description of isolation, shown in Listing 4. The partner web service is executed though invoke activity *PostCodeInvoke*. The result from invocation is replaced with a string constant "TEXAS CITY".
- 4) Isolation of the partner service through transformation of the business process. Figure 2 shows the transformed part



of the business process, where the *Assign1* activity replaces the original *PostCodeInvoke* invoke activity.

Listing 4: Formal description of isolation

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

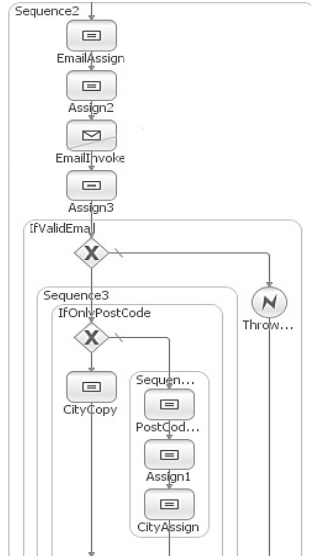


Figure 2: Isolation of partner web service

- 5) Execution of the transformed business process without invocation of *addresslookup* web service.

The output from isolation of partner web service is the same as one presented in Table 1. The partner web service *addresslookup* is successfully isolated without breaking of the normal execution of the business process.

#### 4.3. Fault injection

The robustness testing of the business process is performed through the following steps:

- 1) Selection of BPEL file, containing description of the *Check Payment* business process.
- 2) Formal description of the fault that will be simulated, shown in Listing 5 – a delay in the communication channel between the business process and its partner web service *ValidatorDemo*.

Listing 5: Formal description of fault injection

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

- 3) Transformation of the business process according to the formal description in step 3. The invocation of partner web service *ValidatorDemo* is replaced with an invocation of proxy web service, called *ProxyInvoke*, which injects fault.
- 4) Execution of the transformed business process. The *ProxyInvoke* web service simulates a delay of 20 sec of the response from the partner web service.

The output from fault injection is the same as one presented in Table 1, except the value for the Response time. Due to delay of response from the partner web service *ValidatorDemo*, the actual Response time of the business process is 23718 ms.

#### 5. Summary of methodology validation

TASSA methodology is evaluated based on the following assessment criteria:

- **Applicability** to SOA-based systems;
- **Completeness** of testing related to both WSCs and its components;
- **Level of automation** of the whole testing process;
- **Level of novelty**.

More details about the results from evaluation of TASSA methodology can be found in [62,63,64,65]. They prove that the TASSA methodology provides solution of the *most significant problems* related to the WSC testing: (1) *Inability to instrument web services* that are not under control of the developer; (2) *Delay of testing* due to parallel development of the application components that have dependencies between each other; (3) *Lack automation of testing* for situations when the communication channel is interrupted or there is a noise in it and (4) *Difficulties during identification of the root causes* of failures and lack of a specific analysis approach dealing with different technology platforms.

The validation results shows that the isolation of partner web services can be applied to both synchronous and asynchronous WSCs. The isolation of the external dependencies and the injection of faults keep the original behavior of the WSC. In addition, the proposed approach for fault injection successfully simulate different types of failures. The identification and tracking the execution paths of WSC is supported by TASSA methodology through the approach for dependency analysis. This approach successfully identifies the conditional activities in the BPEL file and the variables, which values determine the execution on a given path.

The evaluation of TASSA methodology according to the above criteria proves its feasibility and effectiveness. The TASSA methodology proposes **a novelty solution for testing SOA-based applications**. Its level of novelty is directly related to the maximum level of novelty of the adopted approaches. Therefore, the level of novelty of TASSA methodology is 3 from total 3. It delivers **a complete solution**. Its completeness is calculated as a sum of completeness of the adopted approaches. The completeness of TASSA methodology is 11 from total 12. The TASSA methodology provides **a high level of automation**. Its level of automation is computed as a sum of level of automation of the approaches that are proposed for its implementation. This metric varies due to possibility for combination of different approaches to achieve a specific testing goal. Since the testing approaches share test data, the manual activities related to its generation on the latest testing stages are minimized. The average level of automation of TASSA methodology is 7 from total 12. It is fully applicable to testing of WSCs, defined with WS-BPEL. The **applicability** of the TASSA methodology is directly related to the minimum applicability of the approaches that are proposed for its implementation. It depends on the concrete approaches used to test a given WSC. When all activities of TASSA methodology are



performed, the applicability is calculated as 1. When the activities related to isolation of partner web services and dependency analysis are not performed, the applicability is calculated as 3. In this case the TASSA methodology can be applied to a number of service-based applications.

Table 2 Summarizes the evaluation results. The first fifth rolls are related to the proposed testing approaches, while the last one shows the results for the TASSA methodology as a whole.

Table 2: Summary evaluation of TASSA methodology

Approach/ methodology	Novelty	Completeness	Automation	Applicability	Total
Isolation of dependencies	100%	100%	66%	25%	73%
Fault injection	66%	100%	66%	75%	77%
Dependencies analysis	100%	33-66%	33-66	25%	48-73%
Functional test case generation	33%	100%	100%	100%	66%
Test data generation	66%	66-100%	66%	75%	68-77%
Overall methodology	100%	80-93%	33-100%	25-100%	66-77%

## 6. Conclusion

The present work addresses the challenges of testing WSCs by proposing a novel methodology for testing, called TASSA. The methodology consists of the following main activities: (1) Checking for compliance with standards and isolation of partner web services; (2) Definition of testing goal and dependency analysis of the WSC to obtain testing paths; (3) Definition of functional test cases; (4) Definition of test cases for performance and load testing; (5) Definition of test cases for security testing; (6) Test case execution; and (7) Recording and reporting of test results. The optional execution of some activities enables focusing of the testing process on the specific user requirements and performing a step-by-step testing. For example, preparation activities could be performed at the beginning of business process development, including checking for compliance with standards. On the next step, when the partner web services are identified and are available, a functional testing could be performed. Finally, when the development of the business process under test is completed, non-functional testing could be conducted. A significant benefit of the TASSA methodology is that it deals with a small set of test artefacts – BPEL file, test data and test assertions. When the methodology is applied to the functional testing, a high level of automation can be achieved, in which only the BPEL file should be provided.

A lot of approaches are explored to identify the most suitable ones for implementation of the activities of TASSA methodology. New approaches are proposed for the most critical issues such as fault injection, isolation of partner services and dependency analysis. For other activities current approaches are adopted and are extended if needed. Two popular approaches for functional test case definition and test data generation are extended. The automation of TASSA methodology is shown onto a sample, yet realistic case study as a proof-of-concept. The complete validation is performed using different business processes in terms of number of activities, number of partner web services and type of communication – synchronous or asynchronous. The novelty, automation and applicability of TASSA methodology to testing

serviced-based applications are evaluated regarding clearly defined assessment criteria. The results shows that it fully support end-to-end testing of WSCs covering all required testing activities.

## Conflict of Interest

The authors declare no conflict of interest.

## Acknowledgment

This research work has been supported by GATE “Big Data for Smart Society” project, funded by the European Union's Horizon 2020 WIDESPREAD-2018-2020 TEAMING Phase 2 programme under grant agreement no. 857155 and Big4Smart “Big Data Innovative Solutions for Smart Cities”, funded by the Bulgarian National Science fund, under agreement no. DN12/9 and agreement no. DN 02/11.

## References

- [1] Petrova-Antonova, S. Ilieva, D. Manova, “TASSA Methodology: End-to-end Testing of Web Service Compositions” in 11th International Conference on the Quality of Information and Communications Technology (QIATIC’2018), September 4-7, 2018, 264-267, Electronic ISBN: 978-1-5386-5841-3. <https://doi.org/10.1109/QUATIC.2018.00046>
- [2] OASIS Web Services Business Process Execution Language, <https://www.oasis-open.org/>, last accessed 2019/11/04
- [3] W3C., <https://www.w3.org/standards>, last accessed 2019/11/04
- [4] T. Lertphumpanya, T. Senivongse, “Basis path test suite and testing process for WS-BPEL” WSEAS Transactions on Computers, 7(5), 483-496, 2008.
- [5] Y. Yuan, Li, Z., W. Sun, “A graph-search based approach to BPEL4WS test generation” in IEEE Int. Conf. on Software Engineering Advances, 2006. <https://doi.org/10.1109/ICSEA.2006.261270>
- [6] IBM - Rational Test Virtualization Server. (2016, January 1). <http://www.ibm.com/software/products/en/rtrvs>, last accessed 2019/11/04
- [7] Web Services API Mocking Overview | SOAP Mocking. (n.d.). <https://www.soapui.org/soap-mocking/service-mocking-overview.html>. last accessed 2019/11/04
- [8] CA Service Virtualization - CA Technologies. (n.d.). <https://www.ca.com/us/products/ca-service-virtualization.html>, last accessed 2019/11/04
- [9] R. Fletcher, “Betamax - Record & replay HTTP traffic”, <http://betamax.software/>, 2011, last accessed 2019/11/04
- [10] Service Virtualization: Application & Data Simulation Software | Hewlett Packard Enterprise. (n.d.). <http://www8.hp.com/us/en/software-solutions/service-virtualization/>, last accessed 2019/11/04
- [11] S. Bruning, S. Weissleder, M. Malek, “A fault taxonomy for service-oriented architecture” in IEEE 10th High Assurance Systems Engineering Symposium, 2007, 367-368. <https://doi.org/10.1109/HASE.2007.46>
- [12] K. M. Chan, J. Bishop, J. Steyn, L. Baresi, S. Guinea, “A fault taxonomy for web service composition” in International Conference on Service-Oriented Computing, 2007, Lecture Notes in Computer Science, vol 4907. Springer, Berlin, Heidelberg, 363-375. [https://doi.org/10.1007/978-3-540-93851-4\\_36](https://doi.org/10.1007/978-3-540-93851-4_36)
- [13] C. Fu, B. G. Ryder, A. Milanova, D. Wonnacott, “Testing of java web services for robustness” ACM SIGSOFT Software Engineering Notes, 29(4), 2004, 23-34. <https://doi.org/10.1145/1007512.1007516>
- [14] M. G. Fugini, B. Pernici, F. Ramoni, “Quality analysis of composed services through fault injection” Inf Syst Front (2009) 11: 227. <https://doi.org/10.1007/s10796-008-9086-3>.
- [15] P. Kumar, Ratneshwer, “A Review on Dependency Analysis of SOA based System” in IEEE Fifth International Conference on Recent Trends in Information, Telecommunication and Computing, 2014, 69-81. <https://doi.org/02.ITC.2014.5.11>
- [16] J. Yan, Z. Li, Y. Yuan, W. Sun, J. Zhang” Bpel4ws unit testing: Test case generation using a concurrent path analysis approach” In IEEE 17th International Symposium on Software Reliability Engineering, 2006, 75–84. <https://doi.org/10.1109/ISSRE.2006.16>
- [17] Z. J. Li, H. F. Tan, H. H. Liu, J. Zhu, N. M. Mitsumori, “Business-process-driven gray-box SOA testing” IBM Systems Journal, 47(3), 2008, 457-472. <https://doi.org/10.1147/sj.473.0457>
- [18] W. L. Dong, H. Yu, Y. B. Zhang, “Testing BPEL-based web service composition using high-level petri nets” in 10th IEEE International Enterprise Distributed Object Computing Conference, 2006. 441-444. <https://doi.org/10.1109/EDOC.2006.59>

- [19] W. M. Van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, E. Verbeek, "Conformance checking of service behavior" *ACM Transactions on Internet Technology*, 8(3), 13, 1-30, 2008. <https://doi.org/10.1145/1361186.1361189>
- [20] M. Daghighzadeh, M. Babamir, "An ABC based approach to test case generation for BPEL processes" in *IEEE 3th International eConference on Computer and Knowledge Engineering*, 2013, 272-277. <https://doi.org/10.1109/ICCKE.2013.6682849>
- [21] H. Cao, S. Ying, D. Du "Towards model-based verification of BPEL with model checking" in *Sixth IEEE International Conference on Computer and Information Technology*, 2006, Seoul, 190-190. <https://doi.org/10.1109/CIT.2006.185>
- [22] Z. Guangquan, R. Mei, Z. Jun, "A business process of web services testing method based on UML2.0 activity diagram" in *IEEE Workshop on Intelligent Information Technology Application*, Zhang Jiajie, 2007, pp. 59-65. <https://doi.org/10.1109/IITA.2007.83>
- [23] C. Bartolini, A Bertolino, E. Marchetti, A. Polini, "WS-TAXI: A WSDL-based testing tool for web services" in *IEEE International Conference on Software Testing Verification and Validation*, 2009, 326-335. <https://doi.org/10.1109/ICST.2009.28>
- [24] C. Ma, C. Du, T. Zhang, F. Hu, X. Cai, "WSDL-based automated test data generation for web service" in *International Conference on Computer Science and Software Engineering*, 2008, 731-737. <https://doi.org/10.1109/CSSE.2008.790>
- [25] H. M. Sneed, S. Huang "WSDLTest-a tool for testing web services" in *Eighth IEEE International Symposium on Web Site Evolution*, 2006, pp. 14-21. <https://doi.org/10.1109/WSE.2006.24>
- [26] J. Offutt, W. Xu "Generating test cases for web services using data perturbation" *ACM SIGSOFT Software Engineering Notes*, 29(5), 2004, 1-1. <https://doi.org/10.1145/1022494.1022529>
- [27] X. Bai, W. Dong, W. T. Tsai, Y. Chen, "WSDL-based automatic test case generation for web services testing" in *IEEE International Workshop on Service-Oriented System Engineering*, 2005, 207-212. <https://doi.org/10.1109/SOSE.2005.43>
- [28] DataGen - XML Test Data Generation Tool (2012). <http://iwm.uni-koblenz.de/datagen/index.html>, last accessed 2019/11/04
- [29] Eclipse XML Editor Help - Eclipse Platform. (n.d.). <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.wst.xmleditor.d oc.user%2Ftopics%2Ffwxmlmledt.html>, last accessed 2019/11/04
- [30] M. Sorens (2009). Taking XML Validation to the Next Level: Introducing CAM. *DevX*. <http://www.devx.com/xml/Article/41066>, last accessed 2019/11/04
- [31] V. Bergmann, *Databene Benerator 0.7.6, manual*, 2012.
- [32] J. Durand, S. D. Green, S. Kulvatunyou, T. Rutt, "Test Assertions on steroids for XML artifacts" in *Balisage: The Markup Conference*, vol. 3, 2009.
- [33] J. Tennison, "Validating XML with Schematron", *Beginning XSLT*, 2004, Apress.
- [34] T. Bacon, J. Martin, "XMLUnit - Unit Testing XML for Java and .NET", <http://www.xmlunit.org/>, last accessed 2019/11/04
- [35] Y. Zheng, J. Zhou, P. Krause "An Automatic Test Case Generation Framework for Web Services" *Journal of Software*, 2(3), 20017, 64-77. <http://epubs.surrey.ac.uk/id/eprint/1975>
- [36] Ch. Ma, J. Wu, Tao Zh., Y. Zhang, X. Cai "Automatic Test Case Generation for BPEL Using Stream X-Machine" *International Journal of u- and e-Service, Science and Technology*, ISSN: 2005-4246, 2008, 27-36.
- [37] Q. Yuan, J. Wu, Ch. Liu, L. Zhang "A model driven approach toward business process test case generation" in *10th International Symposium on Web Site Evolution*, 2008. <https://doi.org/10.1109/WSE.2008.4655394>
- [38] J. Garcia-Fanjul, J. Tuya, Cl. de la Riva "Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN" in *International Workshop on Web Services Modelling and Testing*, 2006, 83-94.
- [39] W. Chen, J. Wei, G. Wu, X. Qiao "Developing a Concurrent Service Orchestration Engine Based on Event-Driven Architecture" *Meersman R., Tari Z. (eds) On the Move to Meaningful Internet Systems: OTM 2008. OTM 2008. Lecture Notes in Computer Science*, vol 5331. Springer, Berlin, Heidelberg, 2008, pp. 61-68.
- [40] Apache ODE Team. *Apache ODE BPEL Engine - Architectural Overview*, 2013. <http://ode.apache.org/developerguide/architectural-overview.html>, last accessed 2019/11/04
- [41] B. Li, X. Y. Tang, J. Lv, "The research and implementation of services discovery agent in web services composition framework". in *IEEE International Conference on Machine Learning and Cybernetics*, 2005, vol. 1, 78-84.
- [42] S. Y. Hwang, H. Wang, J. Tang, J. Srivastava, "A probabilistic approach to modeling and estimating the QoS of web-services-based workflows", *Information Sciences*, 177(23), 2007, 5484-5503.
- [43] N. Kabbani, S. Tilley, "Evaluating the capabilities of SOA security testing tools" in *IEEE International Systems Conference*, 2011, 129-134. <https://doi.org/10.1109/SYSCON.2011.5929125>
- [44] R. Groenboom, R. Jaamour, "Securing Web Services" in *OWASP Europe Conference*, Leuven, Belgium, 2006
- [45] OASIS Web Services Quality Factors Version 1.0. OASIS Committee Specification 01. <http://docs.oasis-open.org/wsqs/WS-Quality-Factors/v1.0/cs01/WS-Quality-Factors-v1.0-cs01.html>, last accessed 2019/11/04
- [46] E. Yuan, J. Tong, "Attributed based access control (ABAC) for web services" in *IEEE International Conference on Web Services*, 2005. <https://doi.org/10.1109/ICWS.2005.25>
- [47] M. Bartoletti P. Degano, G. L. Ferrari, "Plans for service composition" in *Workshop on Issues in the Theory of Security*, 2006.
- [48] S. Haibo, H. Fan, "A context-aware role-based access control model for web services" in *IEEE International Conference on e-Business Engineering*, 2005, pp. 220-223. <https://doi.org/10.1109/ICEBE.2005.1>
- [49] S. M. Babamir, F. S. Babamir S. Karimi, "Design and evaluation of a broker for secure web service composition" in *IEEE International Symposium on Computer Networks and Distributed Systems*, 2011, 222-226. <https://doi.org/10.1109/CNDS.2011.5764577>
- [50] M. Anisetti, C. A. Ardagna, M. Bezzi, E. Damiani, S. P. Kaluvuri, A. Sabetta, "A Certification-Aware Service-Oriented Architecture" *Bouguettaya A., Sheng Q., Daniel F. (eds) Advanced Web Services*. Springer, New York, NY, 2014, 147-170.
- [51] J. Biskup, B. Carminati, E. Ferrari, F. Muller, S. Wortmann, "Towards secure execution orders for composite web services" in *IEEE International Conference on Web Services*, 2007, 489-496.
- [52] B. Zhou, D. Llewellyn-Jones, Q. Shi, M. Asim, M. Merabti, D. Lamb, "A Compose Language-Based Framework for Secure Service Composition" in *IEEE International Conference on Cyber Security*, 2012, 195-202. <https://doi.org/10.1109/CyberSecurity.2012.32>
- [53] F. Martinelli, I. Matteucci, M. Petrocchi, L. Wiegand, "A formal support for collaborative data sharing", in *International Conference on Availability, Reliability, and Security*, Springer Berlin Heidelberg, 2012, 547-561.
- [54] C. K. Yee, *Design and Implementation of Test Case Generation Tool for BPEL Unit Testing* (Doctoral dissertation, thesis), 2007.
- [55] H. Roth, J. Schiefer, A. Schatten, "Probing and monitoring of WSBPEL processes with web services" in *3rd IEEE International Conference on E-Commerce Technology, 8th IEEE International Conference on and Enterprise Computing, E-Commerce, and E-Services*, 2006, 30-30. <https://doi.org/10.1109/CEC-EEE.2006.69>
- [56] L. Baresi, C. Ghezzi, S. Guinea, "Smart monitors for composed services" in *ACM 2nd international conference on Service oriented computing*, 2004, 193-202. <https://doi.org/10.1145/1035167.1035195>
- [57] Active Endpoint, *ActiveBPEL*. [www.active-endpoints.com/active-bpel-engineoverview](http://www.active-endpoints.com/active-bpel-engineoverview), 2007. htm, last accessed 2019/11/04
- [58] L. Jamen, S. Ghosh, "Oracle Fusion Middleware Performance and Tuning Guide 11g" [https://docs.oracle.com/cd/E21764\\_01/core.1111/e10108/bpel.htm#ASPER99175](https://docs.oracle.com/cd/E21764_01/core.1111/e10108/bpel.htm#ASPER99175), last accessed 2019/11/04
- [59] C. Johnson, B. Newport, "Develop high performance J2EE threads with WebSphere Application Server" [http://www.ibm.com/developerworks/websphere/techjournal/0606\\_johnson/0606\\_johnson.html](http://www.ibm.com/developerworks/websphere/techjournal/0606_johnson/0606_johnson.html), last accessed 2019/11/04
- [60] M. Oriol Hilari, J. Marco Gómez, J. Franch Gutiérrez, D. Ameller, "Monitoring adaptable SOA systems using SALMon" in *1st Workshop on Monitoring, Adaptation and Beyond*, 2008, 19-28.
- [61] F. Barbon, P. Traverso, M. Pistore, M. Trainotti, "Run-time monitoring of instances and classes of web service compositions" in *IEEE International Conference on Web Services*, 2006, 63-71. <https://doi.org/10.1109/ICWS.2006.113>
- [62] D. Manova, I. Manova, S. Ilieva, D. Petrova-Antonova, "faultInjector: A Tool for Injection of Faults in Synchronous WS-BPEL processes" in *IEEE 2nd Eastern European Regional Conf. on the Engineering of Computer Based Systems*, 2011, 99-105. <https://doi.org/10.1109/ECBS-EERC.2011.23>
- [63] D. Petrova-Antonova, D. Manova, S. Ilieva, "TASSA: Testing Framework for web service orchestration". in *IEEE/ACM 10th International Workshop on Automation of Software Test*, 2015, 8-12. <https://doi.org/10.1109/AST.2015.9>
- [64] D. Petrova-Antonova, S. Ilieva, D. Manova, "Automated Web Service Composition Testing as a Service" in *Hammoudi S., Pires L., Selic B., Desfray P. (eds) Model-Driven Engineering and Software Development. MODELSWARD 2016. Communications in Computer and Information Science*, vol 692. Springer, Cham, 114-131.
- [65] I. Spassov, D. Petrova, V. Pavlov, S. Ilieva, "Data Dependency Analysis Tool for Web Service Business Processes" *Murgante B., Gervasi O., Iglesias A., Taniar D., Apduhan B.O. (eds) Computational Science and Its Applications - ICCSA 2011. ICCSA 2011. Lecture Notes in Computer Science*, vol 6786. Springer, Berlin, Heidelberg, 232-243.