# Approaches to functional, structural and security SOA testing

**Cesare Bartolini**
*Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Pisa, Italy*
**Antonia Bertolino**
*Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Pisa, Italy*
**Francesca Lonetti**
*Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Pisa, Italy*
**Eda Marchetti**
*Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche, Pisa, Italy*

## ABSTRACT

In this chapter, we provide an overview of recently proposed approaches and tools for functional and structural testing of SOA services. Typically, these two classes of approaches have been considered separately. However, since they focus on different perspectives, they are generally non-conflicting and could be used in a complementary way. Accordingly, we make an attempt at such a combination, briefly showing the approach and some preliminary results of the experimentation. The combined approach provides encouraging results from the point of view of the achievements and the degree of automation obtained. A very important concern in designing and developing web services is security. In the chapter we also discuss the security testing challenges and the currently proposed solutions.

## INTRODUCTION

Traditional testing approaches are divided into two major classes: functional and structural. Functional approaches provide the ability to verify the proper behaviour of services in order to assess and validate their functionality. They treat the applications under test as black boxes, focusing on the externally visible behaviour but ignoring the internal structure. Structural approaches, also known as white-box testing, on the other hand, are a well-known valuable complement to functional ones. Coverage information can provide an indication of the thoroughness of the executed test cases, and can help identify additional test cases to functional ones which exercise unexecuted paths and hence might help detect further faults.
The same division is still valid considering SOA: functional approaches are applied in SOA testing either to show the conformance to a user-provided specification, or according to their fault detection ability, assessed on fault models. In this chapter, we overview some of the existing proposals that derive functional test cases using formal specification, contracts or WSDL language.
Concerning structural testing of SOA, two different points of view can be identified: coverage measured at the level of a service composition (orchestration or choreography) and coverage of a single service . Generally, validation of service orchestrations is based on the Business Process Execution Language description considered as an extended control flow diagram. Classical techniques of structural coverage (e.g., control flow or dataflow) can be used to guide test generation or to assess test coverage so as to take into consideration the peculiarities of the Business Process Execution Language. Other proposals are

instead based on formal specification of the workflows, e.g., Petri Nets and Finite State Processes used for verifying specific service properties.

Considering a service choreography, existing research focuses, among others, on service modeling, process flow modeling, violation detection of properties such as atomicity and resource constraints, and XML-based test derivation.

If, on the one side, there are several approaches for structural testing of service compositions, there are few proposals for deriving structural coverage measures of the invoked services. The reason for this is that independent web services usually provide just an interface, enough to invoke them and develop some general (black-box) tests, but insufficient for a tester to develop an adequate understanding of the integration quality between the application and independent web services. We describe an approach that addresses this deficit by "whitening" services testing through the addition of an intermediate coverage service.

In this chapter, we first provide a survey of some proposed approaches and tools for supporting SOA functional and structural testing. Then, we propose an example of application of a selected black-box approach and a selected white-box approach to a case study for comparative purposes.

In SOA services, another important aspect that must be carefully checked is security, since functional and structural testing, albeit successfully executed, do not prevent security weaknesses.

Different testing strategies, usually divided into passive and active mechanisms, can be used to provide evidence in security-related issues, i.e., that an application meets its requirements in presence of hostile and malicious inputs. We will survey the most commonly-adopted methodologies and techniques such as fuzz testing, injection, and web services security extensions.

An important facet of security information management in web applications is the control of accesses. We will hence include testing methodologies exploiting the specification of access control policies by means of policy languages, such as the eXtensible Access Control Markup Language (XACML) or the Role-Based Access Control (RBAC). We will overview current proposals for access control policy testing dealing with the classification of the possible policy faults and the development of the corresponding fault model, the application of standard or ad hoc conceived test coverage criteria to measure the adequacy of a test suite, and the automated generation of test cases using (for example), change-impact analysis, random heuristics or model-based approaches.

## FUNCTIONAL TESTING

The functional testing of Web Services is a critical issue both in research and in the IT industry. Nowadays business and social welfare are more and more depending on the proper functioning of services delivered over the Net. Therefore, WSs need to be thoroughly tested before deployment. In this section we will provide an overview of the recent proposals and tools for the functional testing of Web Services.

As a common guideline this kind of testing relies on the functionality provided by the Web Services. Commonly the use of the (formal) specification or the XML Schema datatype available allows the generation of test cases for boundary-value analysis, equivalence class testing or random testing. The derived test suite could have different purposes, such as to prove the conformance to a user-provided specification, to show the fault detection ability assessed on fault models, or to verify the interoperability by exploiting the invocation syntax defined in an associated WSDL (WS Description Language) document (WSDL, 2007).

In this last case the formalized WSDL description of service operations and of their input and output parameters can be taken as a reference for black box testing at the service interface.

Considering conformance testing, some of the current proposals include the Coyote framework, proposed by (Tsai, Paul, Song, & Cao, 2002), which requires the user to build test cases as Message Sequence Charts (Harel, & Thiagarajan, 2004). A contract specification of the services under test is also necessary. In (Jiang, Hou, Shan, Zhang, & Xie, 2005) the authors focused on the fault detection ability of the test data, which is assessed using mutation on contracts.

A simpler approach is proposed in (Siblini, & Mansour, 2005), where mutations are defined on the WSDL language. However the mutation operations adopted in both approaches do not correspond to any widely accepted fault-model.
Another approach about testing the conformance of a WS instance to an access protocol described by a UML2.0 Protocol State Machine (PSM) is presented in (Bertolino, Frantzen, Polini, & Tretmans, 2006) and (Bertolino, & Polini 2005). This protocol defines how the service provided by a component can be accessed by a client through its ports and interfaces. The PSM is translated into a Symbolic Transition System (STS), on which existing formal testing theory and tools can be applied for conformance evaluation. For instance, in (Frantzen, & Tretmans, 2006), STSs are used to specify the behavior of communicating WS ports and test data are generated to check the conformity of the effective implementation to such a specification. The advantage of this approach is that it uses a standard notation (UML 2.0 PSM) for the protocol specification.
Based on the use of WSDL and XML Schema datatype information, the WSTD-Gen tool, developed by (Li, Zhu, Zhang, & Mitsumori, 2009), generates the test cases by exploiting the information extracted from the WSDL specifications and user knowledge. The tool implements also the possibility of customizing the data types and the selection of the test generation rules. The peculiarity of this approach is that it is focused on proving the non-existence of known errors in the system according to a fault model used, so erroneous test data are generated intentionally.
WSDL-based test data generation is considered in other proposed approaches such as (Ma, Du, Zhang, Hu, & Cai, 2008; Sneed, & Huang, 2006; Bai, Dong, Tsai, & Chen, 2005; Heckel, & Mariani, 2005), which mainly focus on the definition of test cases for testing a single web service operation.
Some commercial tools for testing WSDL descriptions of Web services are also available such as soapUI (Eviware, n.d.), TestMaker (PushToTest, n.d.), and Parasoft (Parasoft, n.d.). In particular, soapUI is acclaimed on its distribution site as the most used tool for WSs testing. It can automatically produce a skeleton of a WS test case and provide support for its execution and result analysis.
The tester's job is certainly greatly released by the usage of this or similar tools; however the produced test cases are incomplete and lack the input parameter values and the expected outputs. Moreover, soapUI can also measure the coverage of WS operations, but again the generation of diverse test messages for adequately exercising operations and data combinations is left to the human testers.
From our study of the literature, we find it somehow surprising that WS test automation is not pushed further as of today. Indeed, the use of XML-based syntax of WSDL documents could support fully automated WS test generation by means of traditional syntax-based testing approaches. In this direction, we have recently proposed an approach (Bartolini, Bertolino, Marchetti, & Polini, 2009) which is a sort of “turn-key” generation of WS test suites.
This approach relies on a practical yet powerful tool for fully automated generation of WSs test inputs. The key idea is to combine the coverage of WS operations (as provided by soapUI) with well-established strategies for data-driven test input generation. The tool, called WS-TAXI, is obtained by integrating two existing softwares: soapUI, presented above, and TAXI (Bertolino, Gao, Marchetti, & Polini, 2007), which is an application for the automated derivation of XML instances from an XML schema (the interested reader can refer to (Bertolino, Gao, Marchetti, & Polini, 2007) for details on TAXI implementation). The integrated tool is named WS-TAXI.
The original notion at the basis of WS-TAXI, in comparison with soapUI and other existing WS test tools, is the inclusion of a systematic strategy for test generation based on basic well-established principles of the testing discipline, such as equivalence partitioning, boundary analysis and combinatorial testing.
The test cases generated by the approaches or tools surveyed so far are scoped for testing a single operation per time and the possible dependencies among the test cases are not considered. The combination of different test cases according to the behavioral dependencies is considered in (Bai, Dong, Tsai, & Chen, 2005). In particular the use of specifications containing behavioural information is typical of the semantic web service testing or the ontology-based test data generation. The use of semantic

models such as OWL-S for test data generation is proposed for instance in (Bai, Lee, Tsai, & Chen, 2008) while ontology-based test data generation is proposed in (Wang, Bai, Li, & Huang, 2007).

## STRUCTURAL TESTING

As we discussed in the Introduction, most often structural testing of SOA applications is devoted to the validation of Web Service Compositions (WSC). Validation of WSCs has been addressed by some authors suggesting to perform structural coverage testing of a WSC specification. In several approaches, it is assumed that the WSC is provided in BPEL (OASIS WSBPEL, 2007), the Business Process Execution Language, a standard for programming WSCs.

In (Yuan, Li, & Sun, 2006), the BPEL description is abstracted as an extended control flow diagram; paths over this diagram can be used to guide test generation or to assess test coverage. A major issue in this approach comes from the parallelism in BPEL which results in a much more complex control flow and a very high number of paths.

To cope with this problem, some proposals make several simplifying assumptions on BPEL and consider only a subset of the language. In (García-Fanjul, Tuya, & de la Riva, 2006) a transformation is proposed from BPEL to Promela (similarly to (Cao, Ying, & Du, 2006)). The resulting abstract model is used to generate tests guided by structural coverage criteria (e.g. transition coverage). Similar complexity explosion problems may be encountered in such methods, since the amount of states and transitions of the target model can be very high.

Different approaches for conformance testing that exploit model-checking techniques are available. All these works usually focus on the generation of test cases from counterexamples given by the model checker. Considering in particular the usage of SPIN, the work of (García-Fanjul et al. 2006) generates test cases specification for compositions given in BPEL by systematically applying the transition coverage criterion. (Zheng, Zhou, & Krause, 2007) transforms each BPEL activity into automata which are successively transformed into Promela, the input format of the SPIN model checker. Similarly also (Fu, Bultan, & Su, 2004) uses the SPIN model checker to verify BPEL but without passing through Promela as in (García-Fanjul, Tuya, & de la Riva, 2006). In this case in fact the BPEL is translated to guard conditions which are transformed into Promela.

When a formal model of the WSC and of the required properties is provided, a formal proof can be carried out. For instance, Petri Nets can be built from workflows (Narayanan, & McIlraith, 2003) or from BPEL processes (Yang, Tan, Yong, Liu, & Yu, 2006) or from other approaches (Pistore, Marconi, Bertoli, & Traverso, 2005), to verify properties such as reachability. In (Foster, Uchitel, Magee, & Kremer, 2003), the workflow is specified in BPEL and an additional functional specification is provided as a set of Message Sequence Charts (MSC) (Harel & Thiagarajan, 2004). These specifications are translated in the Finite State Processes (FSP) notation. Model-checking is performed to detect execution scenarios allowed in the MSC description and not executable in the workflow, and vice versa. The complexity of the involved models and model-checking algorithms is the main concern that in practice makes these approaches hardly applicable to real-world WSCs.

The above investigations use models of the composition behavior and of properties or scenarios expressing the user expectations (MSCs or state-based properties such as the absence of deadlock): faults manifest themselves as discrepancies between the system behavior and these expectations. An interesting characterization is proposed in (Weiss, Esfandiari, & Luo, 2007) where failures of WSCs are considered as interactions between WSs, similarly to feature interactions in telecommunications services. Interactions are classified for instance as goal conflict or as resource contention.

Orthogonal approaches are based on the transformation of BPEL into the Intermediate Format Language (IF), as presented for instance in (Lallali, Zaidi, & Cavalli, 2008). In these approaches the test cases are generated using the TestGen-IF tool (Cavalli, Montes De Oca, Mallouli, & Lallali, 2008).

More recently, the WSOFT (Web Service composition, Online Testing Framework) (Cao, Felix, & Castane, 2010) consists of a framework which combines together test execution and debug of test cases focused on unit testing. Specifically the WSOFT approach focuses on the testing of a composition in which all partners are simulated by the WSOFT. Timing constraints and synchronous time delays are also considered. In particular, timing constraints are expressed by the TEFSM (Timed Extended Finite State Machines) formal specification. The same authors present in (Cao, Felix, Castane, & Berrada, 2010) a similar framework that automatically generates and executes tests “online” for conformance testing of a composition of Web services described in BPEL. The framework considers unit testing and is based on a timed modeling of BPEL specification.

Considering specifically the verification of the data transformations involved in the WSC execution, few proposals are available. From the modeling point of view, this lack has been outlined in (Marconi, Pistore, & Traverso, 2006) where a model representing the data transformations performed during the execution of a WSC is proposed. This model can be used together with behavioral specifications to automate the WSC process.

In (Bartolini, Bertolino, Marchetti, & Parissis, 2008) the authors focus on testing based on data-related models and schematically settle future research issues on the perspectives opened by dataflow-based validation of WSs. In particular the authors discuss two possible approaches based on dataflow modeling and analysis for the validation of WSCs. Precisely, a methodology for using a dataflow model derived from the requirements specification and a technique focused on a dataflow model extracted from the BPEL description of the composition.

However, in the contest of Service Oriented Architectures (SOAs) where independent web services can be composed with other services to provide richer functionality, interoperability testing becomes a major challenge. Independent web services usually provide just an interface, enough to invoke them and develop some general functional (black-box) tests, but insufficient for a tester to develop an adequate understanding of the integration quality between the application and the independent web services. To address this lack, in (Bartolini, Bertolino, Elbaum, & Marchetti, 2009) the authors proposed a “whitening” approach to make web services more transparent through the addition of an intermediate coverage service. The approach, named Service Oriented Coverage Testing (SOCT), provides a tester with feedback about how a whitened service, called a Testable Service, is exercised.

## COMBINING FUNCTIONAL AND STRUCTURAL APPROACHES

In this section, we aim at applying some of the approaches presented in the paper, to try and evaluate a combined methodology against a sample web service. Since several approaches cover different steps of the testing process, it is possible to use them in a non-contrasting merged approach, getting the benefits of each of them. Specifically, the experiment proposed here explores the coverage analysis of a web service after choosing a functional technique for generating the test suite.

### Applied approaches

In this section we provide some additional details about the applied testing technique. In particular, we consider the already introduced WS-TAXI (Bartolini, Bertolino, Marchetti, & Polini, 2009) and SOCT (Bartolini, Bertolino, Elbaum, & Marchetti, 2009) approaches, and apply them to a common case study. As said, the purpose of SOCT is to mitigate the limitation associated with the on-line testing activity by enabling the application of traditional white-box testing techniques to SOA applications.

The envisioned testing scenario for SOCT is depicted in Figure 1. The traditional actors in SOA testing (Canfora, & Di Penta, 2009) are the service provider, who can test their service before deployment, and the service integrator, who has to test the orchestrated services. The TCov (Bartolini, Bertolino, Elbaum, & Marchetti 2009) service sits between the service developer and the service provider. The TCov provider can be assumed as a trusted provider of services that delivers coverage information to service providers as it is tested by the service developer.

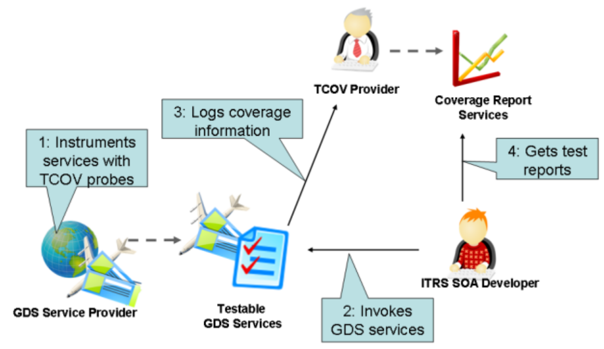

*Figure 1. Overview of the SOCT paradigm.*

To realize the SOCT scenario, the provided services must be instrumented (callout 1 in Figure 1) to enable the collection of coverage data, not differently from how instrumentation is normally performed for traditional white-box testing. As the developer invokes the services during on-line testing (callout 2), coverage measures are collected from services and sent (callout 3) to TCov, which will then be responsible to process the information and make it available to the developer as a service (callout 4).
It is obvious that any means to reveal more of the structure of a service will increase testability. However, SOCT is designed not to reveal anything that would damage an industrial asset of the provider of the instrumented services, because the granularity of the coverage information is a provider's choice.
For generating the test suite, the experiments were carried out with the aid of WS-TAXI, based on the TAXI tool.
TAXI can generate conforming XML instances from an XML Schema. To generate the instances, it uses a modified version of the Category Partition (CP) algorithm (Offutt, & Xu, 2004) to find all possible structures for the elements by adopting a systematic black-box criterion. CP splits the functional specifications of the service into *categories* and further into *choices*. Categories represent the parameters and conditions which are relevant to the algorithm for testing purposes, while choices are the possible values (either valid or invalid) which can be assigned to each category. CP also includes constraints to avoid having redundant tests in the test suite.
In particular, TAXI processes the schema by applying the following rules:

- ***choice*** elements are processed by generating instances with every possible child. Multiple *choice* elements produce a combinatorial number of instances. This ensures that the set of sub-schemas represents all possible structures derivable from *choice;*
- ***Element occurrences*** are analyzed, and the constraints are determined, from the XML Schema definition. Boundary values for *minOccurs* and *maxOccurs* are defined;
- ***all*** elements result in a random sequence of the *all* children elements for generating the instance. This new sequence is then used during the values assignment to each element.

Exploiting the information collected so far and the structure of the (sub)schema, TAXI derives a set of intermediate instances by combining the occurrence values assigned to each element. The final instances are derived from the intermediate ones by assigning values to the various elements. Two approaches can be adopted: values can either be picked randomly from those stored in an associated database, or generated randomly (e.g., a random sequence of characters for a string element) if the database contains no value for an element. Since the number of instances with different structures could be huge, in the current implementation TAXI only selects one value per element for each instance.

## Testbed

The testbed selected for this experimentation was a modified version of WorldTravel (WorldTravel, 2008)[i]. WorldTravel is a web service developed at the Georgia Institute of Technology, specifically to be used as a testbed for research on web services. From a functional point of view, WorldTravel provides research facilities for flights within the United Stated and Canada. The software is written in Java and runs on an Apache Geronimo application server with the Tomcat servlet container.

Although it is designed to run both on Windows and Linux operating systems, only the Linux setup was used for these tests. To retrieve flight information, WorldTravel relies on a MySQL database, which in the current experimentation was running on the same computer as the Geronimo server. The database contains much more data than those actually used by the service.

As previously mentioned, the WorldTravel software was slightly modified. First off, some bugs had to be fixed. Secondly, some improvements were performed with respect to the original version. Changes made included some additional checks on the input data, resulting in error messages if wrong input is provided. Last, the code was instrumented for applying SOCT. Probes were introduced for two different types of coverage analysis, block coverage and branch coverage.

An additional software used for this experimentation is TCov. Technically, it is a simple web service developed in PHP running on an Apache web server and a MySQL database, which collects coverage data for the instrumented web service and allows the user to retrieve those data in summarized reports. In the setup used for this work, TCov resides on a different server from the one running WorldTravel.

Since the point of view of this experimentation is that of the service integrator, i.e., a developer who creates a service composition, another service was created that uses WorldTravel as a backend. This service, which will be referred to as SO (standing for Service Orchestration) from now on, simulates the behavior of a travel agency, with the purpose of allowing customers to check flights availability, according to the trip, the dates, and the desired fare class and/or carrier.

The experiments have been run against this service, which uses WorldTravel and is set to take advantage of the TCov instrumentation to evaluate the coverage reached.

## Test suite generation

The first step consisted in developing the set of test cases to run on the web service. To do this, the web service tester can normally rely on two types of information provided by the web service provider: the functional specification and the WSDL interface. The former contains detailed information on what the web service expects as input data and how these data should be combined. The latter is a machine-readable specification which, among other things, contains the exact structure of the input and output SOAP messages. Although the two pieces of information are quite similar, there are important differences in that the functional specification can be expressed at various degrees of detail, but usually is defined at a higher level, while the WSDL interface contains structural details (ranges, patterns and so on) which are normally not highlighted in the specification.

This difference is reflected in the two different types of functional testing approach which can be applied for test suite generation: one based on the specification, and the other based on the WSDL interface. This led to two different test suites generated for SO, one created through the Category Partition (CP) algorithm, the other using the WS-TAXI tool on the WSDL interface. In the following, the two test suites will be referred to as TS1 and TS2, respectively.

SO receives queries with specific attributes (one-way or round trip, dates, acceptable fare classes, and desired airlines), and finds the best fares among the flights meeting the search criteria. SO has a single operation, *FareResponse bestFare(FareRequest),* which takes the following input data as part of the FareRequest object:

- the originating airport (string)
- the destination airport (string)
- the date of the flight to the destination (dateTime)
- [0..1] the date for the return flight (dateTime), if needed for a round-trip flight

- [0..3] the requested fare classes (enum:  First, Business, Economy)
- [0..*] the requested airlines

The data returned in the *FareResponse* object contains zero or one flight that include:

- [1..2] trips, each containing the flight name, date, origin and destination
- the fare class (string)
- the cost of the flight (int)

The generation of TS1, as mentioned, is based on the aforementioned Category Partition algorithm. For each component of the *FareRequest* object, the following situations be considered: all possible valid structures, missing element, null element, structural error, syntactic error. For example, for the date of the return flight, the following cases were taken into considerations: zero elements, one element with a correct date, one empty element, one element with a date in the wrong format, one element with a date prior to the onward flight. The test suite was generated by making a combinatorial composition of all valid inputs, and by adding single test cases for non-valid inputs. TS1 consists of a total of 31 tests.
TS2, on the other hand, has been generated using WS-TAXI. WS-TAXI is a set of tools which produce and execute a test suite using only the WSDL as an input. The process is the following:

1. the aforementioned soapUI tool is used to generate SOAP envelopes for the invocations of all operations in the WSDL. The non-commercial version of soapUI was used, but only to generate stubs of SOAP calls for the operations declared in a WSDL file. soapUI fills the contents of the envelope with placeholders for data, without assigning significant values. For example, string values are filled with random Latin words, while occurrences are managed simply by introducing a comment in the envelope notifying the minimum and maximum number of elements allowed at that spot, leaving to the user the endeavor to select a specific number of occurrences. Since the purpose of TS2 is to exploit all possible structures for the SOAP calls, this is clearly insufficient, so the envelopes generated by soapUI are completely stripped of their contents, which will be replaced in the following steps. A different software capable of generating SOAP envelopes from WSDL files could be used in alternative, such as Altova XMLSpy (Altova, n.d.);
2. a simple script is used to extract from the WSDL interface all informations related to the data structure of the messages used in the operations. A WSDL generally has a section which contains or refers an XML Schema, and the WSDL messages (*rectius* their parts) contain elements which are described therein. This step produces a separate XSD file named as the WSDL (but obviously with a different extension);
3. the XSD file generated in step 2 is fed as an input to the TAXI tool. For these experiments, the database described previously was populated with the allowed origin and destination airports and the possible carriers; dates were generated randomly, while the fare classes (first, business, and economy) were stored in an enumeration in the schema (TAXI manages enumerations by assigning random values among those allowed). A total of 200 possible instances were generated for the *FareRequest* element;
4. a simple script takes all the XML instances and fills the envelopes created with soapUI with each of them, producing an equal number of SOAP input messages;
5. last, each of the SOAP messages generated in the previous step is sent to the web service, and the response (if it is received and does not time out) is compared against an oracle for correctness.

This process generates a test suite which normally is bigger than the one created with the CP algorithm, but the benefit is that most of the process is automated (except for step 1, since to our best knowledge there is no way to run soapUI from a command line, which would allow to insert it into a batch execution), and requires much less handiwork from the tester. Additionally, there is a greater variability in the values with respect to TS1, and that might be relevant in situations where the behavior of the service is data-dependent.

## Test suites comparison

The two methodologies for building TS1 and TS2 differ a lot under several aspects:

- the effort required to build them: TS1 requires more work on the tester's side, while TS2 is mostly automatic;
- the number of instances generated: TS1 is strictly dependent on the CP algorithm and the granularity of the functional specification (the more detailed and close to the implementation details the specification is, the bigger the number of tests will be), whereas TS2 can produce an arbitrary number of instances (generally a lot more than TS1). At its limit, if the specification is extremely detailed, the two test suites would be made up of the same tests;
- the data variability: TS2 is limited only by the size of the database used by TAXI, while the building of TS1 is not data-driven;
- the structure of the invocations: TS1 also contains non-allowed data structures, which is normally not possible using WS-TAXI;
- the time required for executing the test suite: since TS2 can be much larger, it can take much longer to run all tests on the service.

Obviously, the question that arises is: which one is to be preferred? The answer is not easy, because some of the previous points favor TS1 while others would suggest TS2. The approach chosen to compare the two test suites was to execute each of them in turn and measure the coverage using TCov. Specifically, the incremental coverage was tested, meaning that a single testing session for each test suite was run, and the coverage reached was measured after each test. This is in line with the way TCov is supposed to be used, where a tester would execute tests which used all the features that he needed from the service and then measure how much of the service these features would use.

The results are shown in Figures 2 and 3. The figures display the results both of block and branch coverage. The former shows the coverage measures for TS1, while the latter for TS2.

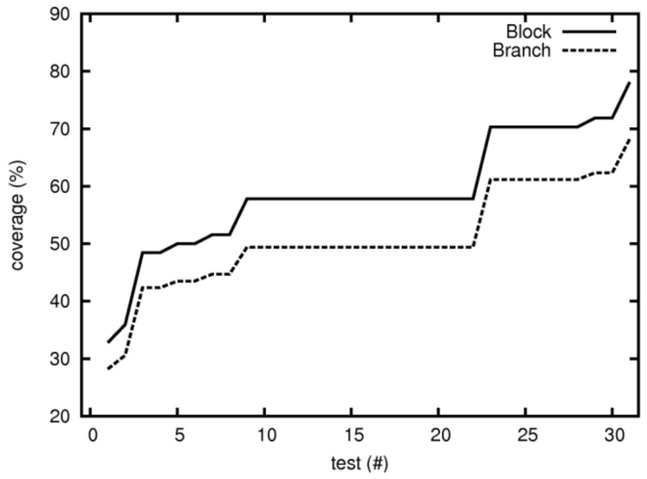

*Figure 2. Coverage measures for TS1.*

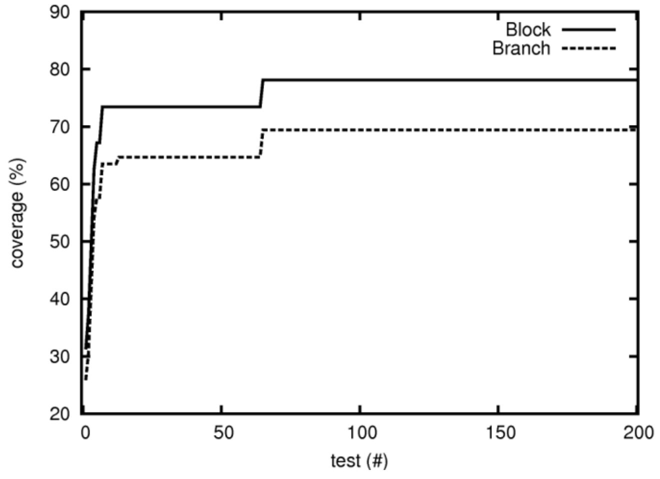

*Figure 3. Coverage measures for TS2.*

The results highlight some important considerations. First off, it can be noted that both test suites eventually reach similar results. This is not unexpected, because both test suites in the end are built using some version of the CP algorithm, plus the behavior of the service is not data-dependent.
TS1 reaches its maximum coverage at the 31st test, measuring 78.125% block coverage and 68.235% branch coverage. TS2, instead, reaches its maximum at the 65th test, with a 78.125% block coverage and 69.412% branch coverage. As expected, TS2 contains a lot more tests which do not exercise new parts of the web service, so many tests are redundant. However, this is not a problem as the test suite is built automatically.
An issue which was not expected at first was that TS2 reached a higher branch coverage than TS1 (albeit slightly). This was considered worthy of closer examination. After some analysis on the detailed coverage results, and comparing those with the source code of WorldTravel, it was discovered that there was a set of input values which TS1 did not exercise whereas TS2 did. This was due to the degree of detail of the functional specification (which didn't produce that specific test in TS1), along with the higher variability in the values offered by TS2.  This occurs because of the way the test suites are built, specifically based on the lack of an information in the functional specifications, and additional runs of each test suite would produce exactly the same coverage. However, the difference is very minor, resulting only in a single probe exercised by TS2 and not by TS1.

## SECURITY TESTING

Security aspects are highly critical in designing and developing web services. It is possible to distinguish at least two kinds of strategies for addressing protective measures of the communication among web services: security at the transport level and security at the message level.
Enforcing the security at the transport level means that the authenticity, integrity, and confidentiality of the message (e.g., the SOAP message) are completely delegated to the lower-level protocols that transport the message itself (e.g., HTTP + TLS/SSL) from the sender to the receiver. Such protocols use public key techniques to authenticate both the end points and agree to a symmetric key, which is then used to encrypt packets over the (transport) connection.
Since SOAP messages may carry vital business information, their integrity and confidentiality need to be preserved, and exchanging SOAP messages in a meaningful and secured manner remains a challenging

part of system integration. Unfortunately, those messages are prone to attacks based on an on-the-fly modification of SOAP messages (XML rewriting attacks or XML injection) that can lead to several consequences such as unauthorized access, disclosure of information, or identity theft. Message-level security within SOAP and web services is addressed in many standards such as WS-Security (OASIS Standard Specification Web Services Security, 2006), which provides mechanisms to ensure end-to-end security and allows to protect some sensitive parts of a SOAP message by means of XML Encryption (Imamura, Dillaway, & Simon, 2002) and XML Signature (Bartel, Boyer, Fox, LaMacchia, & Simon, 2008).

The activity of fault detection is an important aspect of (web) service security. Indeed, most breaches are caused when a system component is used in an unexpected manner. Improperly tested code, executed in a way that the developer did not intend, is often the primary culprit for security vulnerability.

Robustness and other related attributes of web services can be assessed through the testing phase and are designed first off by analyzing WSDL document to know what faults could affect the robustness quality attribute of web services, and secondly by using the fault-based testing techniques to detect such faults (Hanna, & Munro, 2008).

Focusing in particular on testing aspects, the different strategies and approaches that have been developed over the years can be divided into passive or active mechanisms.

Passive mechanisms consist of observing and analyzing messages that the component under test exchanges with its environment (Benharref, Dssouli, Serhani, & Glitho, 2009). In this approach, which has been specially used for fault management in networks (Lee, Netravali, Sabnani, Sugla, & John, 1997) the observer can be either on-line or off-line: an on-line observer collects and checks the exchanged input/output in real time, while an off-line observer uses the log files generated by the component itself. Recently, passive testing has been proposed as a good approach for checking whether a system respects its security policy, as in (Mallouli, Bessayah, Cavalli, & Benameur, 2008; Bhargavan, Fournet, & Gordon, 2008). In this case, formal languages have been used in order to give a verdict about the system conformity with its security requirements.

Active testing is based on the generation and the application of specific test cases in order to detect faults. All the techniques have the purpose of providing evidence in security aspects, i.e., that an application faces its requirements in the presence of hostile and malicious inputs. Like functional testing, security testing relies on what is assumed to be a correct behaviour of the system, and on non-functional requirements. However, the complexity of (web) security testing is bigger than functional testing, and the variety of different aspects that should be taken into consideration during a testing phase implies the use of a variety of techniques and tools.

An important role in software security is played by negative testing (Lyndsay, 2003), i.e., test executions attempting to show that the application does something that it is not supposed to do (Nyman, 2008). Negative tests can discover significant failures, produce strategic information about the model adopted for test case derivation, and provide overall confidence in the quality and security level of the system.

Other common adopted methodologies and techniques include (Wong, & Grzelak, 2006):

- fuzz testing;
- injection;
- policy-based testing.

We will briefly describe validation based on them in the following subsections.

## Fuzz testing

The word fuzzing is conventionally used to refer to a black-box software testing method for identifying vulnerabilities in data handling. It involves generating semivalid data and submitting them in defined input fields or parameters (files, network protocols, API calls, and other targets) in an attempt to break the program and find bugs. Usually the term "fuzz" means feeding a set of inputs taken from random data to a program, and then systematically identifying the failures that arise (Sutton, Greene, & Amini, 2007). Semivalid data are correct enough to keep parsers from immediately dismissing them, but still invalid enough to cause problems. Fuzzing is useful in identifying the presence of common vulnerabilities in data

handling, and the results of fuzzing can be exploited by an attacker to crash or hijack the program using a vulnerable data field. Yet, fuzzing covers a significant portion of negative test cases without forcing the tester to deal with each specific test case for a given boundary condition. Sutton et al. (2007) present a survey of fuzzing techniques and tools.
In particular fuzzing approaches can be divided into: data generation; environment variable and argument fuzzing; web application and server fuzzing; file format fuzzing (Kim, Choi, Lee, & Lee, 2008); network protocol fuzzing, web browser fuzzing; in-memory fuzzing. In the specific area of web service security, fuzzing inputs can often be generated by programmatically analyzing the WSDL or sample SOAP requests and making modifications to the structure and content of valid requests.
File fuzzing strategy is also used for detecting XML data vulnerability. In particular, two main testing methods, generation and mutation, are adopted (Oehlert, 2005). XML-based fuzzing techniques using the first approach parse the XML schema and the XML documents to generate the semivalid data.
The second way to get malformed data is to start with a known set of good data and mutate it in specific places. The authors of (Choi, Kim, & Lee, 2007), propose a methodology and an automated tool performing efficient fuzz testing for text files, such as XML or HTML files, by considering types of values in tags. This tool named TAFT (Tag-Aware text file Fuzz testing Tool) generates text files, extracts tags and analyzes them, makes semivalid data from random data returned by different functions according to types of values and inserts them into values of tags, then automatically executes a target software system using fault-inserted files and observes fault states.

## Injection

In general, web services can interact with a variety of systems and for this reason they must be resistant to injection attacks when external systems are accessed or invoked. The most prevalent injection vulnerabilities include SQL injection, command injection, LDAP injection, XPath injection, and code injection. There are undoubtedly many other forms of injection and the tester should be aware of these for testing different subsystems. Most injection vulnerabilities can be easily and quickly identified by the process of fuzzing due to the presence of meta-characters in various language syntaxes. Recently, SQL Injection Attack has become a major threat to web applications. SQL injection occurs when a database is queried with an SQL statement which contains some user-influenced inputs that are outside the intended parameters range. If this occurs, an attacker may be able to gain control of the database and execute malicious SQL or scripts.
In the following, we will review some solutions to mitigate the risk posed by SQL injection vulnerabilities. The authors of (Huang, Yu, Hang, Tsai, Lee, & Kuo, 2004) secure potential vulnerabilities by combining static analysis with runtime monitoring. Their solution, WebSSARI, statically analyzes source code, finds potential vulnerabilities, and inserts runtime guards into the source code. The authors of (Fu, Lu, Peltsverger, Chen, Qian, & Tao, 2007) present a static analysis framework (called SAFELI) for discovering SQL injection vulnerabilities at compile time. SAFELI analyzes bytecode and relies on string analysis. It employs a new string analysis technique able to handle hybrid constraints that involve boolean, integer, and string variables. Most popular string operations can be handled. The authors of (Halfond, & Orso 2005) secure vulnerable SQL statements by combining static analysis with statement generation and runtime monitoring. Their solution, AMNESIA, uses a model-based approach to detect illegal queries before they are executed on the database. It analyzes a vulnerable SQL statement, generates a generalized statement structure model for the statement, and allows or denies each statement based on a runtime check against the statically-built model.

## Policy-based testing

An important aspect in the security of modern information management systems is the control of accesses. Data and resources must be protected against unauthorized, malicious or improper usage or modification. For this purpose, several standards have been introduced that guarantee authentication and authorization. Among them, the most popular are Security Assertion Markup Language (SAML) (OASIS

Assertions and Protocols for the OASIS Security Assertion Markup Language, 2005) and eXtensible Access Control Markup Language (XACML) (OASIS eXtensible Access Control Markup Language, 2005).

Access control mechanisms verify which users or processes have access to which resources in a system. To facilitate managing and maintaining access control, access control policies are increasingly written in specification languages such as XACML (OASIS eXtensible Access Control Markup Language, 2005), a platform-independent XML-based policy specification language, or RBAC (Role-Based Access Control) (Ferraiolo, Sandhu, Gavrila, Kuhn, & Chandramouli, 2001). Whenever a user requests access to a resource, that request is passed to a software component called Policy Decision Point (PDP). A PDP evaluates the request against the specified access control policies and permits or denies the request accordingly.

Policy-based testing is the testing process to ensure the correctness of policy specifications and implementations. By observing the execution of a policy implementation with a test input (i.e., access request), the testers may identify faults in policy specifications or implementations, and validate whether the corresponding output (i.e., access decision) is as intended. Although policy testing mechanisms vary because there is no single standard way to specify or implement access control policies, in general, the main goals to conduct policy testing are to ensure the correctness of the policy specifications, and the conformance between the policy specifications and implementations.

The authors of (Hu, Martin, Hwang,, & Xie, 2007) classify recent approaches on XACML policy specification testing in the following main categories:

- Fault Models and Mutation Testing: there exist various basic fault models for different types of policies. Martin and Xie (Martin, & Xie, 2007a) propose a fault model to describe simple faults in XACML policies. They categorize faults broadly as syntactic and semantic faults. Syntactic faults are the result of simple typos. For example, in XACML, an XML schema definition (XSD) can be used to check for obvious syntactic flaws. Semantic faults are involved with the logical constructs of the policy language. Based on the fault model, mutation operators are described in (Martin, & Xie, 2007a) to emulate syntactic and semantic faults in policies. The authors of (Le Traon, Mouelhi, & Baudry, 2007) design mutation operators on a given Organization Based Access Control (OrBAC) model. They consider similar semantic and syntactic mutation operators as the preceding ones in addition to other mutation operators including rule deletion, rule addition, and role change based on a role hierarchy.
- Testing criteria: testing criteria are used to determine whether sufficient testing has been conducted and it can be stopped, and measure the degree of adequacy or sufficiency of a test suite. Among testing criteria for policy testing, there are structural coverage criteria and fault coverage criteria (Martin, Xie, & Yu, 2006). The former are defined based on observing whether each individual policy element has been evaluated when a test suite (set of requests) is evaluated by a PDP. The latter are defined based on observing whether each (seeded) potential fault is detected by the test suite.
- Test generation: to test access control policies, policy testers can manually generate test inputs (i.e., requests) to achieve high structural policy coverage and fault coverage (i.e., fault-detection capability). To reduce manual effort, automated test generation approaches can be used. These approaches for test case generation starting by XACML policies deal with random heuristics and test suite reduction (Martin, Xie, & Yu, 2006), policy values based approaches (Martin, & Xie, 2006) and change impact analysis (Martin, & Xie, 2007b). In particular, the Targen tool proposed in (Martin, & Xie, 2006) derives the set of requests satisfying all the possible combinations of truth values of the attribute id-value pairs found in specific sections of a XACML policy. The Cirg tool proposed in (Martin, & Xie, 2007b) is able to exploit change impact analysis for test cases generation starting from policies specification. In particular, it integrates the Margrave tool (Fisler, Krishnamurthi, Meyerovich, & Tschantz, 2005) which performs change impact analysis so as to reach high policy structural coverage. Test generation for model-based policies derives

abstract test cases directly from models such as FSMs that specify model-based policies. The authors of (Le Traon, Mouelhi, & Baudry, 2007) propose test generation techniques to cover rules specified in policies based on the OrBAC (Kalam, Baida, Balbiani, Benferhat, Cuppens, Deswarte, Miège, Saurel, & Trouessin, 2003) model. The same authors in (Mouelhi, Le Traon, & Baudry, 2009) present a new automated approach for selecting and adapting existing functional test cases for security policy testing. The method includes a three-step technique based on mutation applied to security policies and Aspect-Oriented Programming (AOP) for transforming automatically functional test cases into security policy test cases. All the existing solutions for XACML test case derivation are based on the policy specification. Policies, or more general meta-models of security policies, provide a model that specifies the access scheme for the various actors accessing the resources of the system. This has proven effective; however for the purpose of test case generation a second important model composes the access control mechanism: the standard format representing all the possible compliant requests. An approach fully exploiting the potential of this second model has been implemented in the X-CREATE framework (Bertolino, Lonetti, & Marchetti, 2010). Differently from existing solutions that are based only on the policy specification, this framework exploits the special characteristics of XACML access control systems of having a unique and once-for-all specified structure of the input requests: the XACML Context Schema, which establishes the rules to which access requests should conform. This schema is used in the X-CREATE framework for deriving a universally valid conforming test suite that is then customizable to any specific policy.

Finally, we also address some works dealing with policy implementation testing. In particular, the authors of (Li, Hwang, & Xie, 2008) propose an approach to detect defects in XACML implementations by observing the behaviours of different XACML implementations (policy evaluation engine or PDP) for the same test inputs, whereas (Liu, Chen, Hwang, & Xie, 2008) concerns the performance of XACML request processing and presents a scheme for efficient XACML policy implementation.

## CONCLUSION AND FUTURE RESEARCH ISSUES

In this chapter we overviewed the recent approaches for functional and structural testing of web services and web service compositions. We provided also an example of a combined testing strategy. In particular, we have integrated within an automated environment the execution of a test plan based on a functional strategy with the evaluation of its coverage measure. To the best of our knowledge, it is the first attempt of combining functional and structural strategies in a SOA environment. The results obtained showed that it is possible to achieve a coverage measure even starting from a specification-based test plan. As expected, high coverage can be achieved both by generating the test suite by hand and automatically, but the latter requires less time and effort and is more exhaustive. This first attempt paves the way to a new field of research about the integration of different testing strategies.

Finally, we also surveyed approaches to security-based validation. Due to the pervasiveness of web services in business and social life, they should guarantee to preserve privacy and confidentiality of carried-out information.

Thus in the world of Web 2.0, where the distribution of on-line services and applications has reached a high granularity, functional, structural approaches and security testing constitute a third complementary and essential facet of SOA validation. In particular, since most of services involve money transactions, relevant personal information and other critical contexts, they need to be tested thoroughly, and possibly fast for quick deployment.

For this an essential research issue for future research is the creation of an automated testing process, both on the side of test suite generation and steps that the tester must perform to execute the tests. This is especially true in service compositions where, being a high number of applications involved, a single point of failure could make the whole system collapse.

Considering specifically the functional approaches, most of them are still strongly constrained by the need of a formal specification model to be provided along with the service. In many circumstances, companies do not provide such detail and testers are often required to derive it from their own experience. In this

situation, looser approaches based on interfaces (e.g., WSDL) are more suited, yet less effective. A possible field for future research could be the development of strategies located halfway between the two extremes. This way, the specification-based part of the approach would drive test case generation, whereas the interface-based part would drive test case execution.
In particular, a special case of the combined vision could be to use the access control policies as a refinement of the functional specification. This would produce test cases that, before been executed, must be compliant with the specified access policy constraints.
The approach presented in this chapter of combining different methodologies is also an example of a direction where research should progress in the future. Combined methodologies not only help speed up the testing process by allowing simultaneous analysis from multiple perspectives, but can also bring an extra added value by taking the best of the individual approaches.
In this chapter we presented a combination of functional and structural testing, but clearly it is not the only possibility. We plan to investigate different combination strategies, in particular considering the benefits of combining security testing with the other ones.

## ACKNOWLEDGEMENTS

This chapter provides an overview of work that has been partially supported by the European Project FP7 IP 216287: TAS$^3$ and by the Italian MIUR PRIN 2007 Project D-ASAP.

## REFERENCES

Altova. XML Spy. Retrieved from http://www.altova.com/products/xmlspy/xml_editor.html

Bai, X., Dong, W., Tsai, W.T., & Chen, Y. (2005). WSDL-Based Automatic Test Case Generation for Web Services Testing, Proceding of the IEEE International Workshop on Service-Oriented System Engineering (SOSE), Beijing, 207-212.

Bai, X., Lee, S., Tsai, W.T., & Chen, Y. (2008) Ontology-based test modeling and partition testing of web services. Proceedings of the 2008 IEEE International Conference on Web Services. IEEE Computer Society. September, Beijing, China, 465–472.

Bartel, M., Boyer, J., Fox, B., LaMacchia, B., & Simon, E. (2008). *W3C Recommendation XML Signature Syntax and Processing*. Retrieved from http://www.w3.org/TR/xmldsig-core/.

Bartolini, C., Bertolino, A., Elbaum, S.C, & Marchetti, E. (2009). Whitening SOA testing. Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT InternationalSymposium on Foundations of Software Engineering. Amsterdam, The Netherlands, August 24-28, 161-170.

Bartolini, C., Bertolino, A., Marchetti, E., & Parissis, I. (2008) Data Flow-based Validation of Web Services Compositions: Perspectives and Examples, in V, R. de Lemos, F. Di Giandomenico, H. Muccini, C. Gacek, M. Vieira (Eds.). Architecting Dependable Systems, Springer 2008. LNCS5135, 298-325.

Bartolini, C., Bertolino, A., Marchetti, E., & Polini, A. (2009) WS-TAXI: a WSDL-based testing tool for Web Services. Proceeding of 2nd International Conference on Software Testing, Verification, and Validation ICST 2009. Denver, Colorado USA, 2009, April 1- 4.

Benharref, A., Dssouli, R., Serhani, M., & Glitho, R. (2009). Efficient traces´collection mechanisms for passive testing of Web Services. Information and Software Technology, 51(23), 362-374 .

Bertolino, A., & Polini, A. (2005). The audition framework for testing web services interoperability. 31st EUROMICRO International Conference on Software Engineering and Advanced Applications, 134-142.

Bertolino, A., Frantzen, L., Polini, A., & Tretmans, J. (2006). Audition of Web Services for Testing Conformance to Open Specified Protocols. In R. Reussner and J. Stafford and C. Szyperski (Ed). Architecting Systems with Trustworthy Components. (pp. 1-25). LNCS 3938. Springer-Verlag.

Bertolino, A., Gao, J., Marchetti, E., & Polini, A. (2007). Automatic Test Data Generation for XML Schema-based Partition Testing. Proceedings of the Second international Workshop on Automation of Software Test International Conference on Software Engineering. IEEE Computer Society. May 20 - 26, 2007. Washington.

Bertolino, A., Lonetti, F., & Marchetti, E. (2010). Systematic XACML request generation for testing purposes. Accepted for publication to the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). 1-3 September 2010. Lille, France.

Bhargavan, K., Fournet, C., & Gordon, A. (2008). Verifying policy-based web services security In Proc. of the 11th ACM conference on Computer and communications security, 268-277.

Canfora, G., & Di Penta, M. (2009). Service-oriented architectures testing: A survey. Software Engineering Journal, 78-105.

Cao, H., Ying, S., & Du, D. (2006). Towards Model-based Verification of BPEL with Model Checking. In Proceeding of Sixth International Conference on Computer and Information Technology (CIT 2006). 20-22 September 2006, Seoul, Korea, 190-194.

Cao, T.D., Felix, P., & Castane, R. (2010). WSOTF: An Automatic Testing Tool for Web Services Composition. *Proceeding of the Fifth International Conference on Internet and* eb Applications and Services, 7-12.

Cao, T.D., Felix, P., Castane, R. & Berrada, I.. (2010). Online Testing Framework for Web Services, Proceeding of the Third International Conference on Software Testing, Verification and Validation, 363-372.

Cavalli, A., Montes De Oca, E. , Mallouli, W., & Lallali, M. (2008). Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints. Proceeding of 12th IEEE International Symposium on Distributed Simulation and Real TimeApplications, Canada, Oct 27 – 29.

Choi, Y., Kim, H., & Lee, D. (2007). Tag-aware text file fuzz testing for security of a software System. In Proc. of the International Conference on Convergence Information Technology, 2254-259.

Eviware (n.d.) SoapUI. Retrieved from soapUI. http://www.soapui.org/.

Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., & Chandramouli, R. (2001). Proposed NIST standard for rolebased access control. ACM Transactions on Information and System Security, 4(3), 224-274.

Fisler, K., Krishnamurthi, S., Meyerovich, L.A., & Tschantz, M.C.(2005) Verification and change-impact analysis of access-control policies. In Proc. of ICSE 2005, 196-205.

Foster, H., Uchitel, S., Magee, J., & Kremer, J. (2003) Proceeding of Model-based verification of web service compositions. In ASE. IEEE Computer Society, 152-163.

Frantzen, L., & Tretmans, J.(2006). Towards Model-Based Testing of Web Services. International Workshop on Web Services - Modeling and Testing (WS-MaTe2006. Palermo. Italy. June 9$^{th}$, 67-82.

Fu, X., Bultan, T., & Su, J. (2004). Analysis of Interacting BPEL Web Services. Proceeding of International Conference on World Wide Web. May, New York, USA, 17 – 22.

Fu, X., Lu, X., Peltsverger, B., Chen, S., Qian, K., & Tao, L. (2007). A static analysis framework for detecting SQL injection vulnerabilities. In Proc. of COMPSAC, 87-96.

García-Fanjul, J. Tuya, J., & de la Riva, C. (2006). Generating Test Cases Specifications for BPEL Compositions of Web Services Using SPIN. Proceeding of International Workshop on Web Services Modeling and Testing (WS-MaTe2006).

Halfond, W. & Orso, A. (2005). AMNESIA: analysis and monitoring for neutralizing sql injection Attacks. In Proc. of 20th IEEE/ACM International Conference on Automated Software Engineering, 174-183.

Hanna, S., & Munro, M. (2008). Fault-Based Web Services Testing. In Proc. of Fifth International Conference on Information Technology: New Generations 471-476.

Harel, D. & Thiagarajan, P. (2004). Message sequence charts. UML for Real, Springer, 77-105.

Heckel, R. & Mariani, L. (2005). Automatic conformance testing of web services. Fundamental Approaches to Software Engineering. LNCS 3442, 2 - 10 April, 2005, 34-48.

Hu, V. C., Martin, E., Hwang, J., & Xie, T. (2007). Conformance checking of access control policies specified in XACML. In Proc. of the 31st Annual International Conference on Computer Software and Applications, 275-280.

Huang, Y., Yu, F., Hang, C., Tsai, C., Lee, D. & Kuo, S. (2004). Securing web application code by static analysis and runtime protection. In Proc. of 13th International Conference on World Wide Web, 40-52.

Imamura, T., Dillaway, B., & Simon, E. (2002). *W3C Recommendation XML Encryption Syntax and Processing.* Retrieved from http://www.w3.org/TR/xmlenc-core/.

Jiang, Y., Hou, S.S., Shan J. H., Zhang, L., & Xie, B. (Ed.) (2005). Contract-Based Mutation for Testing Components. IEEE International Conference on Software Maintenance.

Kalam, A. A. EI., Baida, R., Balbiani, P. , Benferhat, S. , Cuppens, F., Deswarte, Y. , Miège, A. Saurel, C. , & Trouessin, G. (2003). Organization based access control. In Proc. of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks, 120-131.

Kim, H., Choi, Y., Lee, D., & Lee, D. (2008). Practical Security Testing using File Fuzzing. In Proc. of 10th International Conference on Advanced Communication Technology, 2, 1304-1307.

Lallali, M., Zaidi, F., & Cavalli, A. (2008). Transforming BPEL into Intermediate Format Language for Web Services Composition Testing. Proceeding of 4th IEEE International Conference on Next Generation Web Services Practices, October.

Le Traon, Y., Mouelhi, T., & Baudry, B. (2007). Testing security policies: Going beyond functional testing. In Proc. of the 18th IEEE International Symposium on Software Reliability, 93-102.

Lee, D., Netravali, A., Sabnani, K., Sugla, B., & John, A. (1997). Passive testing and applications to network management. In Proc. of. International Conference on Network Protocols, 113-122.

Li, N., Hwang, J., & Xie, T. (2008). Multiple-implementation testing for XACML implementations. In Proc. of Workshop on Testing, Analysis and Verification of Web Software, 27-33.

Li, Z.J., Zhu, J., Zhang, L.J., & Mitsumori, N. (2009).Towards a Practical and Effective Method for Web Services Test Case Generation. Proceeding of ICSE Workshop on Automation of Software Test AST. May, 106-114.

Liu, A. X., Chen, F., Hwang, J., & Xie, T. (2008). Xengine: A fast and scalable XACML policy evaluation engine. In Proc. of International Conference on Measurement and Modeling of Computer Systems, 265-276.

Lyndsay, J. (2003). A positive view of negative testing. Retrieved from http://www.workroom-productions.com/papers.html

Ma, C.,  Du, C., Zhang, T., Hu, F., & Cai, X. (2008).  WSDL-based automated test data generation for web service. 2008 International Conference on Computer Science and Software Engineering. Wuhan, China, 731-737.

Mallouli, W., Bessayah, F., Cavalli, A., & Benameur, A. (2008). Security Rules Specification and Analysis Based on Passive Testing. In Proc. of IEEE Global Telecommunications Conference, 1-6.

Marconi, A., Pistore, M., & Traverso, P. (2006). Specifying data-flow requirements for the automated composition of web services. Proceeding of the  Fourth IEEE International Conferfeence on Software Engineering and Formal Methods (SEFM 2006), 11-15 September 2006, Pune, India, 147-156 .

Martin, E., & Xie, T.(2006). Automated test generation for access control policies. In Supplemental Proc. of ISSRE

Martin, E., & Xie, T.(2007a). A fault model and mutation testing of access control policies. In Proc. of the 16th International Conference on World Wide Web, 667-676.

Martin, E., & Xie, T.(2007b). Automated test generation for access control policies via change-impact analysis. In Proc. of SESS 2007, 5-11.

Martin, E., Xie, T., & Yu, T. (2006). Defining and measuring policy coverage in testing access control policies. In Proc. of ICICS,  139-158.

Mouelhi, T., Le Traon, Y., & Baudry, B. (2009). Transforming and selecting functional test cases for security policy testing. In Proc. of International Conference on Software Testing Verification and Validation, 171-180.

Narayanan, S., & McIlraith, S. (2003). Analysis and Simulation of Web Services. *Journal of Computer Networks*. 42 (5), 675-693.

Nyman, J. (2008). Positive and Negative Testing GlobalTester, TechQA, 15(5). Retrieved from http://www.sqatester.com/methodology/PositiveandNegativeTesting.htm

OASIS Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. (2005) Retrieved from http://docs.oasis-open.org/security/saml/v2.0/

OASIS eXtensible Access Control Markup Language (XACML) version 2.0. (2005).Retrieved from http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

OASIS Standard Specification Web Services Security. (2006). SOAP Message Security 1.1, (WS-Security 2004), Retrieved from http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

OASIS WSBPEL (2007)  Web Services Business Process Execution Language Version 2.0. Retrieved from http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

Oehlert, P. (2005). Violating assumptions with fuzzing. IEEE Security and Privacy, *3*(2), 58-62.

Offutt, J. & Xu, W. (2004). Generating test cases for web services using data perturbation. SIGSOFT Softw. Eng. Notes, 29(5), 1-10.

Parasoft (n.d.). Retrieved from http://www.parasoft.com/jsp/home.jsp.

Pistore, M.,  Marconi, A., Bertoli, P., & Traverso, P. (2005). Automated Composition of Web Services by Planning at the Knowledge Level. Proceeding of Intenational Joint Conference on Artificial Intelligence (IJCAI). Pp. 1252-1259.

PushToTest (n.d.). TestMaker. Retrieved from http://www.pushtotest.com/.

Sneed, H.M., & Huang, S. (2006). WSDLTest-a tool for testing web services. Proceeding of the IEEE Int. Symposium on Web Site Evolution. Philadelphia , PA, Usa IEEE Computer Society, pp. 12-21.

Sutton, M., Greene, A., & Amini, P. (2007). Fuzzing: Brute Force Vulnerability Discovery.Addison-Wesley

Tsai, W. T., Bai, X., Paul, R., Feng, K., & Yu, L (2002). Scenario-Based Modeling and Its Applications. IEEE WORDS

Tsai, W. T., Paul, R., Song, W., & Cao, Z. (2002). Coyote: an XML-based Framework for Web Services Testing. 7th IEEE International Symp. High Assurance Systems Eng. (HASE 2002).

Wang, Y., Bai, X., Li, J. & Huang, R. (2007). Ontology-based test case generation for testing web services. Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems. Sedona, AZ, USA, Mar. 2007, IEEE Computer Society. 43–50.

Weiss, M, Esfandiari, B., & Luo, Y. (2007). Towards a classification of web service feature interactions. Computer Networks 51(2), 359-381.

Wong, C., & Grzelak, D. (2006). A Web Services Security Testing Framework. SIFT: Information Security Services. Retrieved from http://www.sift.com.au/assets/downloads/SIFT-Web-Services-Security-Testing-Framework-v1-00.pdf.

WorldTravel. (2008). Retrieved from http://www.cc.gatech.edu/systems/projects/WorldTravel

WSDL (2007).Web Services Description Language (WSDL) Version 2.0. Retrieved from http://www.w3.org/TR/wsdl20/.

Yan, J., Li, Z., Yuan, Y., Sun, W. & Zhang, J. (2006). BPEL4WS Unit Testing: Test Case Generation Using a Concurrent Path Analysis Approach. Ptoceeding of 17th International Symposium on Software Reliability Engineering (ISSRE 2006), 7-10 November, Raleigh, North Carolina, USA. 75-84.

Yang, Y, Tan, Q.P., Yong, X., Liu, F., & Yu, J. (2006). Transform BPEL Workflow into Hierarchical CP-Nets to Make Tool Support for Verification. Proceeding of Frontiers of WWW Research and Development - APWeb 2006, 8th Asia-Pacific Web Conference. (pp. 275-284). Harbin, China, January 16-18, 2006.

Yuan, Y., Li, Z., & Sun. (2006). A Graph-Search Based Approach to BPEL4WS Test Generation. In Proceedings of the International Conference on Software Engineering Advances (ICSEA 2006), October 28 – November 2, Papeete, Tahiti, French Polynesia.

Zheng, Y., Zhou, J., & Krause, P. (2007). An Automatic Test Case Generation Framework for Web Services", Journal of Software. Vol 2, No. 3, September

## KEY TERMS & DEFINITIONS

Functional Testing, Structural Testing, Security Testing, XACML, SOA Services.

SOA Testing: The set of testing approaches and methodologies focused on the verification and validation of SOA specific aspects.

Structural Testing: Also called white-box testing, requires complete access to the object's structure and internal data, which means the visibility of the source code.

Functional Testing: Also called black-box testing, relies on the input/output behaviour of the system.

Security Testing: Verification of security-related aspects of the application.

[i] The modified version of WorldTravel can be requested to the authors with an email to cesare.bartolini@isti.cnr.it.