Testing Web Services: A Survey

Technical report TR-10-01

Mustafa Bozkurt, Mark Harman and Youssef Hassoun
Centre for Research on Evolution, Search & Testing
King's College London
Strand, London WC2R 2LS, UK
{mustafa.bozkurt,mark.harman,youssef.hassoun}@kcl.ac.uk

Abstract

The Service-Oriented Computing (SOC) paradigm is allowing computer systems to interact with each other in new ways. According to the literature, SOC allows composition of distributed applications free from their platform and thus reduces the cost of such compositions and makes them easier and faster to develop. Currently web services are the most widely accepted service technology due to the level of autonomy and platform-independency they provide. However, web services also bring challenges. For example, testing web services at the client side is not as straightforward as testing traditional software due to the complex nature of web services and the absence of source code. This paper surveys the previous work undertaken on web service testing, showing the strengths and weaknesses of current web service testing strategies and identifying issues for future work.

1 Introduction

This paper presents a survey of web service testing techniques. Web services is a rapidly growing concept that drives the Service-Oriented Computing (SOC) at present. Web services present important challenges to software testers. These challanges has led to much work on techniques for testing web services. The present paper seeks to provide a comprehensive survey of existing work.

According to Papazoglou [110], SOC is a new computing paradigm that utilizes services as the lightweight constructs to support the development of rapid, low-cost and easy composition of distributed applications. This is a widely used definition of SOC.

The concept of a "service" is comparatively elusive and consequently harder to describe. The definition adopted for this survey is:

> *Services are autonomous, platform-independent computational elements that can be described, published, discovered, orchestrated and programmed using standard protocols to build networks of collaborating applications distributed within and across organizational boundaries [111].*

This definition describes services in terms of Service-Oriented Architecture (SOA) and as a result the definition captures service's technological aspects. In order to justify this description, the terms used need to be explained a little further.

According to the Oxford English Dictionary [109], the meaning of autonomy is:

autonomy (ô-tŏn'-mē) n. pl. autonomies

1. Of a state, institution, etc.: The right of self-government, of making its own laws and administering its own affairs. (Sometimes limited by the adjs. local, administrative, when the self-government is only partial; thus English boroughs have a local autonomy, the former British colonies had an administrative autonomy; political autonomy is national independence.)
   (a) Liberty to follow one's will, personal freedom.
   (b) Metaph. Freedom (of the will); the Kantian doctrine of the Will giving itself its own law, apart from any object willed; opposed to heteronomy.
2. Biol. Autonomous condition:
   (a) The condition of being controlled only by its own laws, and not subject to any higher one.
   (b) Organic independence.
3. A self-governing community (cf. a monarchy).

As this definition suggests, autonomy includes self-government, self-control, independence, self-containment, and freedom from external control and constraint. As a result there are different definitions of autonomy in software. One of these definitions that suits the autonomy of services comes from Erl. According to Erl [44], autonomy, in relation to software, is a quality that "represents the independence with which a program can carry out its logic".

For services, Erl defines autonomy at two levels; runtime and design-time. The runtime autonomy is the level of control a service has over its processing logic at the time of its invocation. The goal behind this autonomy is to increase runtime performance, reliability and behaviour predictability. Increase in all these aspects of services increases reusability of services. Design-time autonomy is the level of freedom service providers have to make changes to a service over its lifetime. This autonomy allows scalability.

Other sources [11, 32, 88] also define autonomy of services in a manner similar to Erl. As an example, Bechara [11] claims that services need to be autonomous in the sense that their operation is independent from other co-operating services. According to Bechara this kind of autonomy is needed to enforce reusability of services.

Services need to be platform-independent in order to provide high reusability. This, in turn, requires interoperability among services. Platform-independency in this context means that the service user must be able to use the functions provided by the service regardless of the platform upon which the user operates. This kind of platform-independent usage of a service requires platform-independent description of the service and platform-independent messaging among the service and its users.

The last part of the definition describes services in terms of SOA usage. In SOA, services must be described, published, discovered and orchestrated in order to perform their functions within the SOA environment. The communication between the service and its user also needs to use standard protocols in order to ensure interoperability.

The reason for having different definitions for services lies in the context in which services are used. For example some researchers [26, 53, 84, 123] define services from the context of GRID Computing and their definition focus is on the aspects of GRID technology. On the other hand Jones [68] defines services in business terms and claims that defining a service from solely a technological point of view is insufficient.

According to Jones, a service description must include other aspects that cannot be measured or defined purely by technology alone.

A recent survey by Canfora and Di Penta [23] summarizes the testing of web services and categorises the research undertaken into four categories: functional, regression, integration and non-functional testing. The current survey extends Canfora and Di Penta's survey by classifying research undertaken in testing web services according to the testing techniques used and by including research areas not covered in Canfora and Di Penta's survey such as formal verification of web services and other more recent work.

The remainder of this survey is organized as follows. Section 2 briefly explains the SOC paradigm, SOA and web services and the underlying technologies in order to make the survey self-contained. Section 3 discusses the issues related to testing web services. Section 4 explains test case generation techniques that are applied to web services. Section 5 examines partition testing of web services. Section 6 discusses the issues related to unit testing of web services and proposed solutions to these issues. Section 7 examines model-based testing and formal verification of web services. Section 8 discusses contract-based testing of web services. Section 9 discusses fault-based testing of web services. Section 10 introduces the collaborative testing concept and describes approaches that use this concept. Section 11 discusses regression testing of web services. Section 12 discusses interoperability testing of web services. Section 13 discusses integration testing of web services. Section 14 concludes the survey.

2 Background

This section introduces the concepts and technologies that are relevant to web services and web service testing.

2.1 Service-Oriented Computing

SOC shifts the traditional understanding of software application design, delivery and consumption. The idea of SOC is that it ought to be able to create a more systematic and a more efficient way of building distributed applications. The vision underpinning this idea is to establish a world of loosely coupled services, able to rapidly assemble dynamic business processes and applications.

The characteristics of SOC applications are claimed to deliver advantages over the traditional distributed applications. Such advantages include platform independency, autonomy and dynamic discovery and composition. There are two primary characteristics of SOC applications through which these advantages occur [150]. These characteristics are:

1. In SOC, all the services must comply with the interface standards so that services are guaranteed to be platform-independent.

2. Service descriptions must enable the automation of integration process (search, discovery and dynamic composition).

Several authors have claimed that the focus of businesses is migrating from product manufacturing (hardware and software) to service provision. For example, according to AppLabs [3], in the future, business concentration will shift away from system development towards core business. One motivation for this shift is the ability to build systems dynamically using services provided by other businesses. According to the International Data Corporation (IDC) [64], a leading market research body, the global spend on service-oriented software in 2006 was nearly $2 billion. The same report projects that this spend will rise to $14 billion in 2011.

In order to achieve effective SOC, integration of many technologies and concepts from different disciplines within software engineering will be required. Of course, this integration will bring numerous challenges as well as advantages. Some of these challenges require adaptation of the existing solutions to SOC and the others need new solutions.

One of the barriers to enterprises' transition to SOC systems is denoted by the heightened importance of the issue of trust. This issue has many dimensions such as correct functioning of service, service security and also Quality of Service (QoS).

Testing provides one potential approach to the issue of trust. Testing is important to assure the correct functioning of service-oriented systems that, by nature, have the ability to dynamically select and use services. In order to confirm the correct functioning of a service-oriented system, interoperability among all its components and integration of these components must be adequately tested.

SOC may also require more frequent testing than traditional software. The possibility of changes to a service-oriented system increases with the number of services involved. With every change to a service, the service-oriented system needs to be tested. Testing service-oriented systems for changes presents other challenges, such as when to test for changes and which operations are affected by the changes. These problems occur when changes are made to services of other businesses.

Because of the reasons stated above, service-oriented systems require more effective and efficient testing than traditional software systems. Unfortunately, most of the existing testing approaches for distributed systems are not adequate enough or applicable to SOC due to the limitations SOC brings. According to Canfora and Di Penta [21], the issues that limit the testability of service-oriented applications are:

1. Limitations in observability of service code and structure due to users having access to service's interface only.
2. Lack of control due to independent infrastructure on which services run and due to provider being the only control mechanism over service evolution.
3. Dynamicity and adaptiveness that limit the ability of the tester to determine the web services that are invoked during the execution of a workflow.
4. Cost of testing:
   (a) The cost of using a service (for services with access quotas or per-use basis).
   (b) Service disruptions that might be caused by massive testing.
   (c) Effects of testing in some systems, such as stock-exchange systems, where each usage of the service means a business transaction.

Specific testbeds are also required. These testbeds must be able to perform as many of the required testing methods as possible and also must be easy to deploy. Ease of deployment is important because it usually takes a long time to set a testing environment for distributed systems. However, the delays may not be acceptable in SOC testing due to SOC's requirement for increased test frequency.

2.2 Service-Oriented Architecture (SOA)

SOA is described as a strategy for building service-oriented applications. Its aim is to provide services that can be used by other services. In SOA, there are three main participants; a service provider, a service user and a service broker. These three participants perform the three fundamental SOA actions; publish,

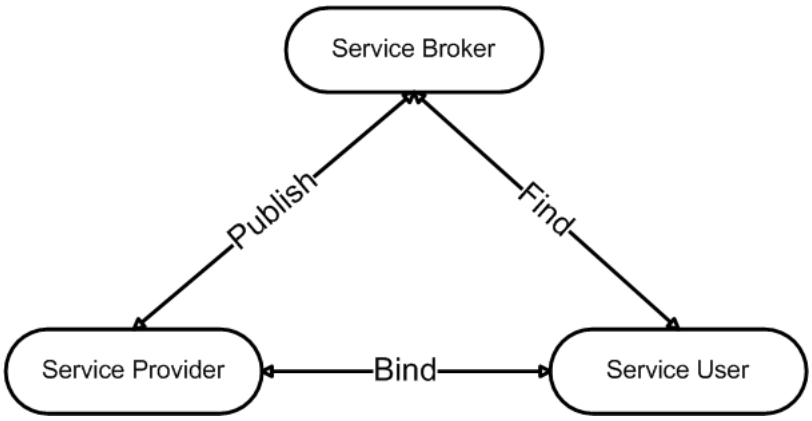

Figure 1. Service-Oriented Architecture

find and bind. Figure 1 illustrates this foundational SOA concept, its participants and the operations among participants.

The service provider is the owner of the service and is responsible for solving service problems and service maintenance. The service provider is also the sole controller of service evolution. The host on which the service resides can also be accepted as the provider in terms of architecture. The service provider publishes a service by registering it to a service broker.

The service broker can be seen as a lookup mechanism for services. It is a registry in which services are published and using which, searches can be performed. It allows users to find services that match their requirements and provides information on how to access these services (details used for binding).

The service user is the most important component since it initiates the two major operations; find and bind. After finding a service that satisfies his needs, the service user invokes the service with the binding information from the broker. This binding information includes the location of the service, how to access the service and the functions that are provided by the service.

2.3 Web Services and Web Service Technologies

A web service is defined as "a software system designed to support interoperable machine-to-machine interaction over a network" by W3C (World Wide Web Consortium) [170]. The aim of the web service platform is to provide the required level of interoperability among different applications using pre-defined web standards. The web service integration model is loosely coupled in order to enable the required flexibility in integration of heterogeneous systems.

There are different web service styles such as Representational State Transfer (REST) web services and Simple Object Access Protocol (SOAP) web services. They are all based on the SOA but differ in the interfaces that they use. For example SOAP web services use SOAP interface to carry messages and WSDL to describe the services, whereas REST web service interfaces are limited to HTML using common HTTP methods (GET, DELETE, POST and PUT) to describe, publish and consume resources. This survey focuses on the SOAP web services (referred to as web service(s) hereafter) due to their popularity both in industry and academia.

The idea of SOA is older than that of web services but the 'great leap' of SOA has been facilitated with

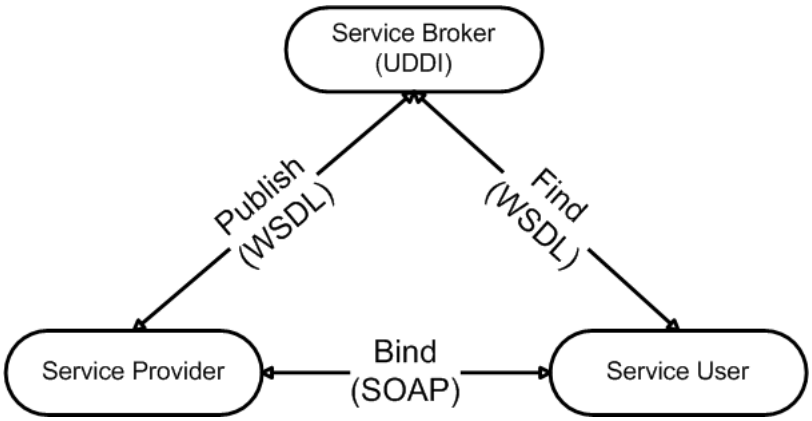

Figure 2. Web Service Architecture

the introduction of web services. Web services are generally accepted as the core element of SOA, providing the necessary autonomy, platform-independence and dynamic discovery and composition. Through web services, pre-existing systems can exchange information without the need to know any technical information about the other computer system. Figure 2 describes the web service architecture and its core specifications that are used in performing each SOA operation. It also illustrates the way in which web services implement the general concept of SOA.

In order to provide platform-independent messaging across a network, the web service architecture uses three core specifications:

1. *Simple Object Access Protocol (SOAP)*: SOAP is an XML-based protocol that allows data exchange over the Hypertext Transfer Protocol (HTTP). The W3C definition of SOAP is "a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment" [139]. Since SOAP as a protocol combines platform independent XML and HTTP, SOAP messages can be exchanged between applications regardless of their platform or programming language. The SOAP protocol allows exchange of XML messages with structured information between a web service and its users.

2. *Web Service Description Language (WSDL)*: W3C defines WSDL as "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information" [169]. A WSDL specification is a web service interface that provides users with all the information they need, such as message formats, operations provided by the web service and location of the web service, to interact with the service in the pre-defined standards.

3. *Universal Description Discovery and Integration (UDDI)*: UDDI is defined as "a set of services supporting the description and discovery of businesses, organizations, and other Web services providers, the web services they make available, and the technical interfaces which may be used to access those services" by OASIS (The Organization for the Advancement of Structured Information Standards) [157]. UDDI is an industry initiative that enables businesses to publish their services and allows potential users to discover these services. UDDI registries can be public or private and users can search and select

a suitable service from a UDDI registry. A web service is not required to be registered to any UDDI registry in order to be used.

There are also other web service technologies that enable composition of multiple web services such as the Business Process Execution Language (BPEL) and the semantic web service specifications that work towards automated integration in SOA.

BPEL is an executable language that defines and manages business processes that involve multiple web services [112]. BPEL is an XML-based flow language for composing multiple web services. Using BPEL, organizations can automate their business processes by orchestrating services within their network and the services from other businesses. The ability to use services from other organizations allows enterprises to build complex processes that include multiple organizations.

Unfortunately, traditional web services (WSDL services) provide only syntactic interfaces and current UDDI brokers support only index word-based searching of web services with required attributes. Using traditional web services, automated discovery and invocation is almost impossible. In order to achieve automation, service descriptions must be machine understandable. Semantic web services have emerged aiming to solve the problems regarding automation.

The aim of semantic web service technology is to provide a richer semantic specification for web services. At present there are several semantic web service proposals that include initiatives and projects such as WSDL-S [165], OWL-S [108], WSMO [164], METEOR-S [97] and SWSA/SWSL [131].

3 Testing Web Services

> *"Web services are not yet widely used because of security concerns. But there's an even bigger roadblock waiting just down the road – **it's called trust**. The big issue is 'Will the service work correctly every time when I need it?' As yet few are thinking about the issues of testing and certification. We suggest that testing and certification of Web services is not business as usual and that new solutions are needed to provide assurance that services can really be trusted."* CBDI Forum,2002 [24].

Testing web services at the client side is one of the major problems that slows down the wider use of web services. The increased attention to SOC and web services is also reflected on their testing.

Testing web services includes testing of the basic web service functionality, web service interoperability, some of the SOA functionalities, QoS and load/stress testing [154]. Canfora and Di Penta [22] discussed the challenges involved in testing different aspects of web services and categorized these challenges. In the present survey the term "user-side issues" will be used to refer to the challenges ascribed to the integrator, third-party and user.

According to Bloomberg [19], the history of web service testing is divided into three phases, based on the functionalities that are added to web service testing tools during these periods:

1. During phase one (2002-2003) web services were considered to be units and testing was performed in a unit testing fashion using web service specifications.

2. During phase two (2003-2005) testing of SOA and its capabilities emerged. The testing in this phase included testing the publishing, finding, binding capabilities of web services, the asynchronous web service messaging capability and the SOAP intermediary capability of SOA. Testing for QoS also emerged as a topic of interest during this period.

3. During phase three (2004 and beyond) the dynamic runtime capabilities of web services were tested. Testing web service compositions and web service versioning testing emerged during this period.

Web service testing is accepted to be more challenging compared to the testing of traditional systems according to the literature. It has been claimed that web services bring new challenges to software testing [21, 5, 145]. These challenges include:

1. Specification based testing using all or some of the web service specifications such as WSDL, OWL-S and WSMO.
2. Runtime testing of web service activities (discovery, binding, publishing) due to the dynamic nature of web services.
3. Testing in collaboration due to the multiple parties involved in web service activities.
4. Testing for web service selection which requires testing web services with same specifications but different implementations.

Testing web services is challenging, not because it brings testing issues that did not exist for traditional software. Testing web services is more challenging compared to traditional systems for two primary reasons; the complex nature of web services and the limitations that occur due to the nature of SOA. It has been argued [5, 52] that the distributed nature of web services based on multiple protocols such as UDDI and SOAP, together with the limited system information provided with WSDL specifications, makes web service testing challenging. The issues that limit the testability of service-oriented applications discussed in Section 2.1, such as limited control and ability to observe, contribute to the testing challenge by reducing the effectiveness of existing testing approaches or by rendering those approaches infeasible or inapplicable.

Despite the challenges that web services bring to software testing, some of the existing testing techniques can be adapted to web service testing [20]. Adaptation is mainly needed to overcome the factors that limit the capability of testing techniques such as not having access to the source code, dynamic nature of web services and having limited control over web services. The amount of changes to a testing technique for its adaptation varies according to the level of control and observation the testing technique requires. In some cases where the testing technique can be directly applied such as black-box unit testing, no adaptation is required. On the other hand, for testing techniques such as regression testing and fault-based testing, only a few of the existing approaches can be used for testing web services and these approaches must be adapted to the web service environment and all its elements.

4 Test Case Generation

According to IEEE standards, a test case is "a documentation specifying inputs, predicted results, and a set of execution conditions for a test item". As the definition states, there are three required elements in a test case; test inputs also referred to as test data, predicted results and conditions of execution. IEEE's definition also clarifies the difference between test data and test case.

In order to generate test cases, test data must first be generated. Test data can be generated using different sources that describe the SUT, SUT's behaviour or how SUT will be used, such as source code, system specifications, system requirements, business scenarios and use cases. Test case generation techniques are usually named after the source that test cases are derived from, such as specification-based or model-based.

In this survey existing work on test case generation is categorized according to the test data generation methods. The two common test data generation methods used in web service testing are specification-based and model-based test data generation.

4.1 Specification-Based Test Data Generation

Specification-based testing is the verification of the SUT against a reference document such as a user interface description, a design specification, a requirements list, a published model or a user manual. Naturally, in specification-based testing, test cases are generated using the available system specifications.

In web service environment the only information a user receives about a web service is its specifications. In this situation specification-based testing becomes a natural choice. Test case generation for web services, as expected, is based on the web service specifications. As explained in Section 2.3, for traditional web services the provided specifications include abstract information on the available operations and its parameters. Information from the specifications allows generation of test cases for boundary-value analysis, equivalence class testing or random testing using the XML Schema datatype information.

Mainly proposed approaches [5, 10, 14, 55, 81, 89, 105, 137] for WSDL-based data generation are based on the XML Schema datatype information. The datatype information with various constraints for each data type allows generation of test data for each simple type. In XML complex datatypes can also be defined. Test data generation for complex datatypes simply requires decomposition of the complex type into simple types. Then test data is generated for each of these simple types and the combined data is used as complex test data.

Li et al. [81] propose a test case generation method that combines information from WSDL specifications and user knowledge. Li et al. introduce a tool called WSTD-Gen that supports this method. The tool allows users to customize data types and select test generation rules for each datatype.

Test cases are not only used for detecting the faults in system behaviour but used for proving the non-existence of known errors. Fault-based test data generation aims to prove the absence of prescribed faults in web services. Unlike regular test data generation, in fault-based test data generation erroneous test data is generated intentionally. As in Offutt and Xu's [105, 176] approach, fault-based test data can also be generated using XML Schema datatypes. Test data generation for fault-based testing is discussed in detail in Section 9.

Proposed approaches [89, 105, 137] for WSDL-based test data generation mostly generate test cases for testing a single web service operation. Test cases that test a single operation might work for testing most web services; however, there are cases that might require test cases that run multiple methods. An example of this situation is an operation that requires the execution of another operation, such as login, as a precondition. Bai et al. [5] address this problem by using data dependencies among the provided operations. The mapping of dependencies is based on the input and output messages of different methods.

Test data generation using WSDL definitions is limited to the datatypes due to the lack of behavioural information about the service. As a result, many researchers look for other alternative specifications that can provide more behavioural information, such as contracts and semantic service specifications. For this matter, the use of semantic web service specifications becomes the natural choice, since they contain more information compared to WSDL. The use of semantic model OWL-S for test data generation is proposed [6, 33, 155, 161] not only because of the behavioural information but also because of the semantic information on the data types. This semantic information in the form of ontology allows ontology-based test data generation [161].

4.2 Model-Based Test Data Generation

Model-based testing is a technique for generating test data from requirements [35]. In model-based testing, test data are derived from a model that describes the SUT's expected behaviour, such as system requirements or specifications. There are also some software development techniques where models are

built before or parallel to the software development process, such as model-driven development. Several formal verification methods using models, such as model-checking, theorem proving and Petri-Nets can also be accepted as model-based testing approaches. These methods are explained in detail in Section 7.

Application of these testing methods to web services is also proposed [39, 51, 69, 77, 135, 136] and most of these proposed approaches generate test cases. For example García-Fanjul et al. [51] generate test cases for BPEL processes using model checking. Test case generation using theorem proving is proposed by Sinha and Paradkar [135]. In this approach web services that work with persistent data are tested using Extended Finite State Machines (EFSMs). Dong and Yu [39] combine High level Petri-Nets (HPN) with constraint-based test data generation for better fault coverage.

There are also other approaches such as Paradkar et al. [113] that use models for test case generation. Paradkar et al. propose a model-based test data generation for semantic web services. In this approach test cases are generated using pre-defined fault-models and IOPE information from semantic specification.

Lallai et al. [71, 72] propose the use of an automata called Web Service Time Extended Finite State Machine (WS-TEFSM) and Intermediate Format (IF) in order to generate timed test cases that aim to exercise the time constraints in web service compositions. An IF model, which enables modelling of time constraints in BPEL, is an instantiation of the WS-TEFSM. For IF model transformation from BPEL Lallai et al. use a tool called BPEL2IF and for test generation another tool called TESTGen-IF. TESTGen-IF is capable of generating test cases for the IF Language. The IF and WS-TEFSM can both model event and faults handlers and termination of BPEL process.

Ma et al. [90] propose the application of Stream X-machine based testing techniques to BPEL. Stream X-machine [74] is a formal method that can model the data and the control of a system and it can be used to generate test cases.

Guangquan et al. [54] propose the use of UML2.0 Activity Diagram [118], a model used for modelling workflows in systems, to model BPEL processes. After modelling the BPEL process, a depth first search method combined with the test coverage criteria is performed on the model in order to generate test cases.

Test cases can be generated using existing data such as existing test cases and recorded user data. This method is proposed for testing web applications and also adapted to web service testing. Conroy et al. [29] generate test cases using the data from applications with Graphical User Interfaces (GUIs). This approach harnesses data from GUI elements and uses the harnessed data to generate test cases for service-oriented systems.

As already discussed, web service testing is considered as black-box testing but BPEL testing, on the contrary, is performed in a white-box manner [79, 92]. When testing BPEL the tester has access to the internal logic of the whole process. As a result, the tester can generate test cases according to the required coverage criteria.

One of the proposed approaches for test case generation to test BPEL processes is based on Graph Search Algorithms (GSA). Since generating Control Flow Graphs (CFGs) from BPEL processes is feasible, this approach is adopted by researchers of this topic [43, 180, 185]. In this approach, test cases are generated from test paths that are created by applying GSAs to CFGs of BPEL processes. The test data for these paths are created by applying constraint solvers.

The approaches using the CFG method mainly propose extensions to standard CFG in order to provide a better representation of BPEL processes. For example, Yan et al. [180] propose an automated test data generation framework that uses an extended CFG called Extended Control Flow Graph (XCFG) to represent BPEL processes. XCFG edges contain BPEL activities and also maintain the execution of activities. Similarly, Yuan et al. [185] propose a graph based test data generation by defining another extended CFG called BPEL Flow Graph (BFG). The BFG contains both the structural information (control and data flow) of a BPEL process that is used for test data generation and semantic information such as dead paths.

Endo et al. [43] propose the use of the CFG approach in order to provide a coverage measure for the existing test sets. Endo et al. propose a new graph called the Parallel Control Flow Graph (PCFG) that contains multiple CFGs representing a service composition and communications among these CFGs and also present a tool that supports the proposed technique called ValiBPEL.

Hou et al. [59] address the issues of message-sequence generation for BPEL compositions. In this approach BPEL processes are modeled as a Message Sequence Graph (MSG) and test cases are generated using this MSG.

Search-based test data generation has attracted a lot of recent attention [1, 93]. Search-based test data generation techniques enable automation of the test generation process, thus reducing the cost of testing. Blanco et al. [17] propose the use of scatter search, a metaheuristic technique, to generate test cases for BPEL compositions. Scatter search works with a population of solutions. In this approach, BPEL compositions are represented as state graphs and test cases generated according to transition coverage criterion. The global transition coverage goal is divided into subgoals that aim to find the test cases that reach the required transitions.

5 Partition Testing

Partition testing is a testing technique that aims to find subsets of the test cases (from existing test cases) that can adequately test a system. The aim of partition testing is to divide input domain of the SUT into subdomains, so that selecting or generating a number of test cases from each subdomain will be enough for testing the entire domain. In essence, partition testing is much like mutation testing, or sometimes mutation testing is considered a partition testing technique [172].

The application of partition testing to web services is proposed at two different levels. Bertolino et al. [14] propose the use of the category-partition method [172] with XML Schemas in order to perform XML-based partition testing. This approach automates the generation of test data using XML Schemas. Bertolino et al. introduce a tool that supports this approach called TAXI. Another approach is proposed by Bai et al. [6] for OWL-S semantic services. Bai et al. introduce a test ontology model that specifies the test concepts and serves as a test contract. Data partitions in this approach are created using the ontology information.

According to Bai et al.'s experiments, using partitioning 76% reduction is achieved; reducing 1413 randomly generated test data for 17 partitions to 344. Comparing Bai et al.'s partition technique against random generation also shows the effectiveness of this approach. In order to cover the 1550 lines of code used in experiments, 60 randomly generated test cases are needed but 20 test cases that were selected using partitioning achieved the same coverage.

The results show that partition testing can help in solving one of the major problems of testing high cost [22] by reducing the required number of test cases. Reduction in the number of test cases will reduce the number of test runs, thereby reducing the cost of testing.

6 Unit Testing of Web Services

Unit testing can be considered the most basic and natural testing technique applicable to any system. In unit testing, individual units of a system that can be independently executed are regarded as units. In terms of web services, the operations provided by a service can be considered as units to be tested.

Unit testing of web services is performed by sending and receiving SOAP messages by the tester. The tester generates the SOAP messages for the operation under test using the information from the WSDL file. In this way, unit testing can verify both the correctness of the WSDL and the correct functioning of the SUT.

Abstract operation information provided by WSDL and details of SOAP protocol are explained in Section 2.3.

According to Canfora and Di Penta [22] one of the main problems of functional testing of web services is its high cost. In testing, the best way of reducing the cost is automation and for unit testing there are existing tools that provide automated testing such as Parasoft SOAtest [140], SOAP Sonar [138], HP service Test [61] and Oracle Application Testing Suite [106]. Even though these tools do help in reducing the manual labor required for test case generation and reporting, they do not fully automate the testing process. In using all these tools, test cases are generated by the tester and the tool generates the SOAP requests for each test case. In some of these tools, even verification of test results has to be performed manually such as in SOAtest. From the provided functionality of all tools, one can assume that automation is not at a desired level.

Fortunately, the need for tools that can automate unit testing was addressed by the research community. For example, Sneed and Huang [137] introduce a tool called WSDLTest for automated unit testing. WSDLTest is capable of generating random requests from WSDL schemata. WSDLTest is also capable of verifying the results of test cases. This capability is achieved by inserting pre-conditions and assertions in test scripts that are manually generated by the tester. The provided verification method requires the tester to be familiar with the SUT in order to generate necessary assertions.

Lenz et al. [77] propose a model-driven testing framework that can perform unit tests. In this approach, JUnit tests are generated using the requirement specifications and the platform-independent tests specifications based on the UML 2 Testing Platform. Both of these required specifications are provided by the service provider.

One of the main problems of software testing is the oracle problem [133, 146]. After the generation of test cases in order to complete the verification process, often a test oracle is needed. An oracle is a mechanism that is used for determining the expected output for each test input. In web service testing, the service user often does not have any reliable test oracles. The lack of a test oracle is one of the challenges of automated web service testing.

Test oracle generation is addressed by Chan et al. [25]. Chan et al. propose a metamorphic testing framework that is capable of performing unit testing. Metamorphic testing [27] can solve the test oracle problem by using metamorphic relations. These relations are defined by the tester for each test suit. Chan et al. also propose the use of metamorphic services that encapsulates a service and imitates the functionalities of the actual service. Verification for the SUT is provided by the encapsulating metamorphic service that verifies the input and the output messages against the metamorphic relations. Chen et al.'s framework enables web service users to determine the expected results but requires the use of relations that can be costly for the provider.

Similarly Heckel and Lochmann [57] generate test oracles using pre-generated contracts. The contracts created using the Design by Contract (DbC) approach are supplied by the service provider and carry information such as pre and post-conditions.

Atkinson et al. [4] propose the use of a technique called *test sheets* in order to generate unit test cases and test oracles. The test sheets approach uses tables defining test cases similar to the Framework for Integrated Test (FIT) [48], a framework for writing acceptance tests. Two types of test sheets are used in this approach; an input test sheet that contains specifications defining a set of test cases and a result test sheet that contains outputs from SUT for the test cases in the test sheets. Atkinson et al. also include contract information that identifies the relation between the operations of a service, defining their effects from the clients' perspective in order to help validation.

An automated solution to the oracle problem is proposed by Tsai et al. [149]. Tsai et al. propose the adaptation of blood group testing to web services and call this technique Adaptive Service Testing and Ranking with Automated oracle generation and test case Ranking (ASTRAR) [147]. ASTRAR is similar to

n-version testing where multiple web services that have the same business logic, internal states and input data are tested together with the same test suite. Even though the main goal of group testing is to test multiple web services at one time to reduce the cost of testing and increase the efficiency, it also helps in solving the reliable test oracle problem within its testing process. ASTRAR can be applied at two different levels of testing: unit and integration testing.

The ASTRAR testing process is divided into two phases; a fast testing phase called the training phase and a hierarchical testing phase called the volume testing phase. For the initial training phase it is assumed that there is no test oracle available but a high number of unranked test cases are available. This phase starts with the selection of a number of randomly selected web services. After the selection of web services, group testing is performed within the training phase by testing selected web services with given test cases. Group testing is followed by a voting mechanism that helps with failure detection and oracle creation. The voting mechanism aims to solve the test oracle problem statistically by analysing the outputs of the web services in group testing. During execution of the voting mechanism, the reliability of web services is calculated and test cases are ranked according to their ability to reveal faults.

Unit testing of web service compositions using BPEL is also addressed. According to Mayer and Lübke [92], BPEL unit testing is performed in two ways; simulated testing and real-life testing. In simulated testing, as opposed to real-life testing, BPEL processes are run on an engine and contacted through a test API instead of regular deployment and invocation. In BPEL testing, web service stubs or mocks can be used instead of the web services that participate in the process. Mayer and Lübke [92] propose a framework that is capable of performing real-world unit testing. This framework can replace participating web services with service mocks. The framework also provides a solution to the asynchronous messaging by providing an implementation of WS-Addressing [166]. Li et al. [79] adopt a different approach for unit testing where BPEL processes are represented as a composition model. Similar to Mayer and Lübke's framework, Li et al.'s framework uses stub processes called *Test Process* to simulate the parts that are under development or inaccessible during the testing process.

Unit testing is one of the most important testing techniques that every system must undergo. The main challenge faced in unit testing of web services is the high cost of testing which can be minimized by automating the testing process. Most of the testing approaches explained in this section provide automated test data generation and test execution, though they lack automated test oracle generation. Tsai et al. [149], Chen et al. [190] and Heckel and Lochmann [57] address this problem and Tsai et al.'s approach provides fully automated test oracle generation.

7 Model-Based Testing and Formal Verification of Web Services

Model-based testing is a testing technique where test cases are generated using a model that describes the behaviour of the SUT. Advantages of model-based testing, such as automating the test case generation process and the ability to analyse the quality of product statically makes it a popular testing technique. The formal and precise nature of modelling also allows activities such as program proof, precondition analysis, model checking, and other forms of formal verification that increase the level of confidence in software [42].

Formal verification of web service compositions is popular due to formal verification methods' ability to investigate behavioural properties. The earliest work on formal verification of web service compositions dates back to 2002. The existing work that has been undertaken in formal verification of web service compositions is compared by Yang et al. [181]. Yang et al.'s work contains the research undertaken between 2002 and 2004. Morimoto [100] surveyed the work undertaken in formal verification of BPEL processes and categorized proposed approaches according to the formal model used. This survey categorizes the work undertaken after 2004 in model-based testing (including formal verification) of web services according to

the testing technique used.

7.1 Model-Based Testing Using Symbolic Execution

Symbolic execution is used as a basis for a verification technique that lies between formal and informal verification according to King [70]. In symbolic testing, the SUT is executed symbolically using a set of classes of inputs instead of a set of test inputs. A class of inputs, represented as a symbol, represents a set of possible input values. The output of a symbolic execution is generated in the form of a function of the input symbols. An example method that generates test cases using symbolic execution is the BZ-Testing-Tools (BZ-TT) method [76]. The BZ-TT method takes B, Z and Statechart specifications as inputs and performs on them several testing strategies, such as partition analysis, cause-effect testing, boundary-value testing and domain testing and several model coverage criteria testing, such as multiple condition boundary coverage and transition coverage. BZ-TT uses a custom constraint solver to perform symbolic execution on the input model.

Testing web services using symbolic execution is proposed by Sinha and Paradkar [135]. Sinha and Paradkar propose an EFSM based approach to test the functional conformance of services that operate on persistent data. In this approach, EFSMs are generated by transforming a WSDL-S model into an EFSM representation. Sinha and Paradkar propose the use of four different test data generation techniques; full predicate coverage, mutation-based, projection coverage and BZ-TT method. For the full predicate coverage each condition of the EFSM is transformed into Disjunctive Normal Form (DNF) [38] and test sequences covering each transition are generated. For the mutation-based test data generation the boolean relational operator method is applied to the guard condition of each operation. For the projection coverage technique the user specifies the test objectives by including or excluding constraints. Sinha and Paradkar's approach is the only approach that performs testing using WSDL-S specifications.

Bentakouk et al. [13] propose another approach that uses symbolic execution in order to test web service compositions. In this approach, web service composition is first translated into a Symbolic Transition System (STS), then a Symbolic Execution Tree (SET) is created using STS of the composition. Bentakouk et al.'s approach takes coverage criteria from the tester and generates the set of execution paths on SET. These generated paths are executed using a test oracle over the service composition. Bentakouk et al. claim that using symbolic execution helps to avoid state-explosion, over-approximation and unimplementable test case problems that are caused by labeled transition systems. As a result, Bentakouk et al.'s approach can handle rich XML-based datatypes.

7.2 Model-Based Testing Using Model-Checking

Model-checking is a formal verification method and is described as a technique for verifying finite state concurrent systems according to Clarke et al. [28]. Model checking verifies whether the system model can satisfy the given properties in the form of a temporal logic. In order to accomplish model checking, a tool called model checker is used. During the proofing process, the model checker detects witnesses and counterexamples for the provided properties. Witnesses and counterexamples are paths in the execution model. A witness is a path where the property is satisfied, whereas a counterexample is a path (sequence of inputs) that takes the finite state system from its initial state to the state where the property is violated. Counterexamples are used for test case generation.

Automaton, referred to as Finite State Automaton or Finite State Machine (FSM) [47], is one of the mathematical models for representing finite state systems. In BPEL testing, automaton models are used for transformation purposes. BPEL processes are first transformed into automata models; these models

are usually transformed into input languages of model-checking tools such as SPIN [141], NuSMV [104] or BLAST [18].

Fu et al. [50] propose a method that uses the SPIN model-checking tool. The proposed framework translates BPEL into an XPath-based guarded automata model (guards represented as XPath expressions [177]) enhanced with unbounded queues for incoming messages. After this transformation, the generated automata model can be transformed into Promela (Process or Protocol Meta Language) specifications [119] with bounded queues (directly) or with synchronous communication based on the result of the synchronizability analysis. The SPIN tool takes Promela as the input language and verifies its LTL (Linear Temporal Logic) [8] properties. Interactions of the peers (participating individual web services) of a composite web service are modeled as conversations and LTL is used for expressing the properties of these conversations.

García-Fanjul et al. [51] use a similar method as Fu et al. [50] in order to generate test cases. García-Fanjul et al.'s approach differs from Fu et al.'s approach in transforming BPEL directly to Promela. García-Fanjul et al. generate test cases using the test case specifications created from counterexamples which are obtained from model-checking. LTL properties are generated for these test case specifications, aiming to cover the transitions identified in the input. Transition coverage for a test suit is achieved by repeatedly executing the SPIN tool with a different LTL formula that is constructed to cover a transition in each execution.

Zheng et al. [189] propose another test case generation method using model-checking for web service compositions. Test coverage criteria such as state coverage, transition coverage and all-du-path coverage are included in the temporal logic in order to perform control-flow and data-flow testing on BPEL. Zheng et al. also propose an automata called Web Service Automaton (WSA) [188] that is used to transform BPEL into Promela for SPIN or SMV for NuSMV model-checker. WSA aims to include BPEL data dependencies that cannot be represented in other automata. This approach generates test cases using counterexamples to perform conformance tests on BPEL and using WSDL to test web service operations.

Huang et al. [62] propose the application of model-checking to semantic web service compositions using OWL-S. The proposed approach converts OWL-S specifications into a C like language and the Planning Domain Definition Language (PDDL) for use with the BLAST model checker. Using BLAST negative and positive test cases can also be generated. Huang et al. propose an extension to the OWL-S specifications and the PDDL in order to support this approach and use a modified version of the BLAST tool.

Betin-Can and Bultan [16] propose the use of model-checking to verify the interoperability of web services. In this approach, it is assumed that the peers of a composite web service are modeled using a Hierarchical State Machines (HSM) based model. Betin-Can and Bultan propose a modular verification approach using Java PathFinder [65], a Java model checker, to perform interface verification, SPIN for behaviour verification and synchronizability analysis.

Ramsokul and Sowmya [121] propose the modelling and verification of web service protocols via model-checking. Ramsokul and Sowmya propose a distributed modelling language based on the novel Asynchronous Extended Hierarchical Automata (ASEHA), which is designed for modelling functional aspects of the web service protocols. ASEHA model of web service protocols are translated into Promela and correctness of the protocol is verified by the SPIN tool.

7.3 Model-Based Testing Using Petri-Nets

Petri-Net (Place/Transition Net) is a mathematical and graphical modelling technique for specifying and analysing concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic systems [101]. Petri-Nets allow different analyses on the model such as reachability, boundness, deadlock, liveness, reversibility, fairness and conservation analysis. Petri-Nets can also be used for measuring test case coverage.

Petri-Nets are also used for model-based testing of web services. For example, Dong and Yu [39] propose

a Petri-Net based testing approach where High level Petri-Net (HPN) [58], a flavour of Petri-Nets, are constructed from WSDL files. This approach uses the generated HPNs for high fault-coverage. Test cases are generated using HPNs and constraint-based test data generation. User-defined constraints for XML datatypes and policy-based constraints specified by the tester provide the necessary constraints for test data generation.

Wang et al. [161] propose the generation of test cases using Petri-Nets and ontology reasoning. In this approach Petri-Net models that are used in describing the operational semantics of a web service are generated from the OWL-S process model and test data is generated using ontology reasoning.

Formal verification of BPEL processes using Petri-Nets has been investigated by Ouyang et al. [107]. Ouyang et al. propose a method for transforming BPEL processes to Petri-Nets with formal analysis of BPEL processes using Petri-Nets models. Two tools are used to automate the transformation and analysis; BPEL2PNML [173] and WofBPEL [173]. BPEL2PNML is a tool that generates the Petri-Net model and WofBPEL is a tool that performs static analysis on Petri-Net models. WofBPEL is capable of checking for unreachable BPEL activities and competition problems for inbound messages.

Schlingloff et al. [129] propose a Petri-Nets-based model-checking approach using the Lola model-checking tool [130] to verify BPEL processes. Schlingloff et al. propose a method called usability analysis to verify the expected behaviour of participating web services.

Lohmann et al. [86] address the problem of analysing the interaction between BPEL processes using a special class of Petri-Nets called Open WorkFlow Net (oWFN). Lohmann et al. introduce two tools that support this approach called BPEL2oWFN that transforms BPEL to oWFN or Petri-Net. oWFN model is used by another tool called Fiona that analyses the interactional behaviour of oWFNs. Petri-Net models are used with model checkers to verify the internal behaviour of a BPEL process.

Different flavours of Petri-Nets are also used in modelling BPEL due to their more expressive nature. For example, Yang et al.'s [181] approach is one of the earliest works that propose the use of Colored Petri-Nets (CP-Nets) [**?**], an extended Petri-Nets, for modelling BPEL processes. Yang et al. list the capabilities of CP-Nets that allow different levels of verifications of BPEL processes such as reachability, boundness, dead transition, dead marking, liveness, home, fairness and conservation analysis. The proposed framework uses CPNTools [31] for CP-Nets analysis and can also verify the BPEL to CP-Net transformation.

Yi et al. [182] also propose a BPEL verification framework that uses CP-Nets. Yi et al. claim that CP-Nets are more expressive than FSM and Petri-Nets and propose the use of CP-Nets in order to model the specifications of the web service conversation protocols. The proposed framework can also help with composition of new BPEL processes and verify the existing processes.

Dong et al. [40] propose the use of HPNs. The proposed approach uses a modified version of a tool called Poses++ which is also developed by Dong et al. The tool is used for automated translation from BPEL to HPN and is also capable of generating test cases.

Dai et al. [34] propose the use of Timed Predicate Petri-Nets (TPPN) with annotated BPEL processes. The proposed annotations are not included in the BPEL itself but they are introduced in annotation layers. These annotations include constraints on the properties of the BPEL process. Dai et al. claim that these annotations can be used in verification of non-functional properties of BPEL as well. This is supported by a tool called MCT4WS that allows automated verifications for web service compositions.

Xu et al. [178] propose the use of the Synchronized-Net model, a model based on Petri-Nets. For this approach Xu et al. use a transformation tool called BPEL2PNML capable of transforming BPEL into Petri-Net Markup Language (PNML) [116]. PNML is used as the input to the Synchronized-Net verification tool.

Similar to Petri-Nets, other models are also used for coverage analysis in model-based testing. For example, Li et al. [78] propose a model-based approach for test coverage analysis. The proposed approach

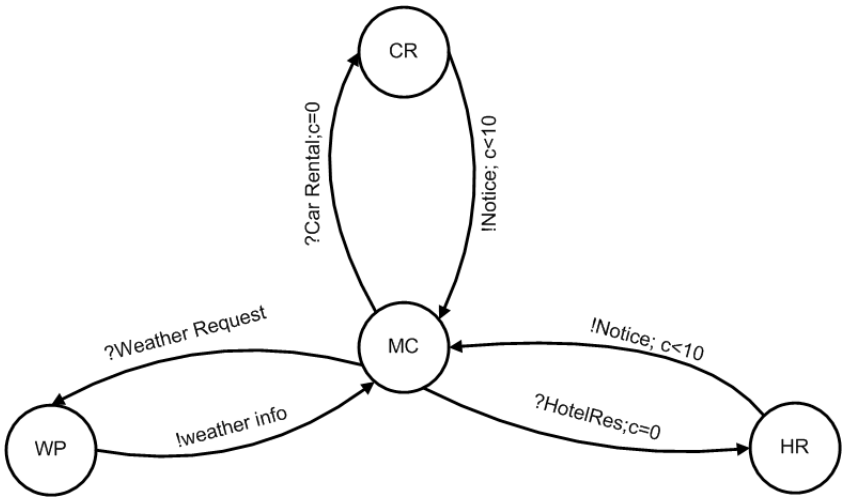

Figure 3. Tarhini et al.'s example TPG of a travel agency application [143]

uses Resource Description Framework (RDF) [124], a language for representing information about resources in the WWW, for specifying the preconditions and effects of each method in WSDL specifications. The preconditions and effects specify, respectively, the state of the service before and after invoking a method.

Tarhini et al. [143] propose two new models for describing composite web services. In this approach SUT is represented with two abstract models; *Task Precedence Graph* (TPG) and *Timed Labeled Transition System* (TLTS). TPG models the interaction between services and TLTS models the internal behaviour of the participating web services. In Figure 3 each node represents a single web service, and edges that join the nodes represent the flow of actions between the two joined nodes. Edges in the model are labeled with the action that needs to be performed and also a time constraint that specifies the response time for each action. If the response time exceeds the specified time limit, the web service that coordinates the flow of actions requests a response from another service for the same action. In Figure 4 each node represents the possible states of a participating web service and edges are the transitions between states. Similar to TPG, each transition is labeled with an action and a time constraint. Tarhini et al. propose three different sets of test case generation for testing different levels of web service composition. Test cases in the first set aim to perform boundary value testing using the specifications derived from the WSDL file. The second set is used for testing the behaviour of a selected web service.The third set tests the interaction between all the participating services and test cases for this set are generated using TPG and TLTS.

Felderer et al. [46] propose a model-driven testing framework called Telling TestStories (TTS). TTS enables tests-driven requirements testing for service-oriented systems. Felderer et al. introduce a domain specific language that allows formalisation of system requirements and test model and test cases to be specified based on the concepts of requirements. Two models are defined in TTS framework; the system model which describes system requirements at business level and the test model that contains test case specifications. TTS framework can also ensure the quality of the test artefacts.

Model-based testing is a popular testing technique. Model-based testing tests a system with a model that sufficiently describes the expected behaviour of the SUT for testing purposes. For BPEL compositions, a model that can sufficiently represent the process can be created thereby allowing model-based testing and

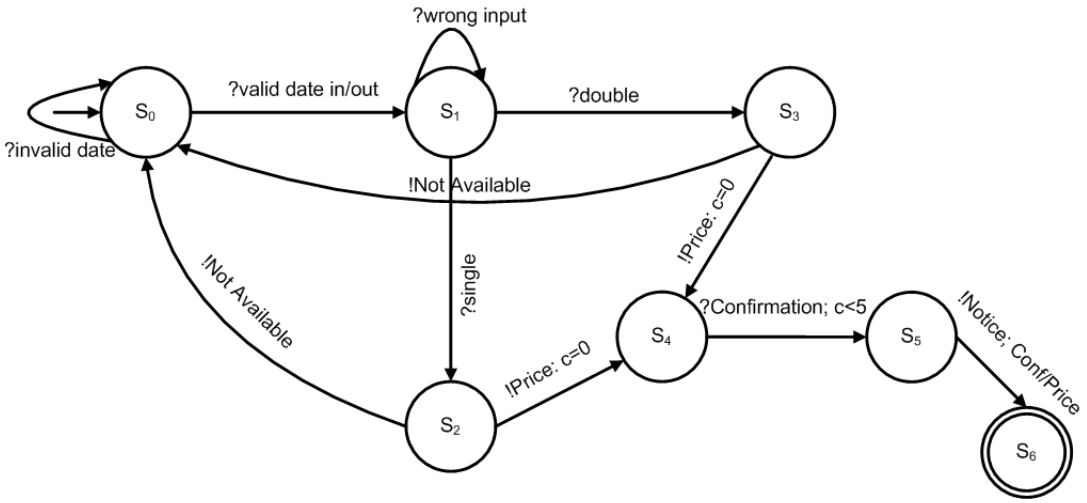

Figure 4. Tarhini et al.'s example TLTS of a hotel reservation [143]

formal verification to be applied. Formal verification of workflows is a widely investigated subject and the popularity of this subject is also reflected on the amount of work produced on the formal verification of BPEL. Unfortunately, a model based on WSDL specifications might not represent the complete behaviour of a web service due to the lack of behavioural information. On the other hand, semantic web service specifications, such as OWL-S and WSDL-S provide more behavioural information and allow a better model of the SUT to be built thus enabling more effective model-based testing.

8 Contract-Based Testing of Web Services

DbC [98] is a software development approach where contracts define the conditions (pre-conditions) for a component to be accessed and the conditions (post-conditions) that need to hold after the execution of methods of that component with the specified pre-conditions. Using contracts, some unexpected behaviour of the SUT can be detected and the information from contracts can also be used to enhance the testing process itself. Software testing using contracts has been applied to traditional software by many researchers [67, 82, 80, 103].

Since traditional web services only provide interface information, researchers have proposed the use of contracts to be used in different aspects of SOA such as service selection, service composition and service verification. These contracts carry information on different aspects of SOA such as behaviour of services and QoS. This extra information on the behaviour of a service such as pre and post-conditions of operations increases the testability of services. This kind of information is invaluable for testing, and testing web services using contracts is investigated by many researchers.

For example, Heckel and Lochmann [57] propose the use of the DbC approach for web services and discuss the reasons for contracts being implemented at different levels such as implementation-level, XML-level and model-level. The contracts defined at model-level are derived from model-level specifications and the reason for this is to minimize the effort required to generate contracts. The created contracts are later used in unit testing of services to check if the service conforms to its specifications. Using contracts,

the proposed testing approach enables automated creation of test cases and test oracles.

Atkinson et al. [4] whose work is discussed in Section 6 propose using a technique called test sheets in order to generate unit test cases and test oracles. Test sheets contain contract information which identifies the relation between the operations of a service. The included relations define the effects of each operation from the clients perspective in order to help validation.

Some of the researchers proposed extensions to WSDL that allow WSDL files to accommodate contract information such as pre and post-conditions. For example, Tsai et al. [153] discuss the reasons for an extension to WSDL in order to perform black-box testing and propose an extended WSDL that carries additional information such as input-output dependency, invocation sequence, hierarchical functional description and sequence specifications. Similarly Mei and Zhang [94] propose an extended WSDL that includes contract information for the service and also a framework that uses this extended WSDL to test services. Heckel and Lochmann's [57] XML-level contracts require an extension to WSDL.

Dai et al. [33] propose contracts that can be contained in OWL-S process model. Proposed contracts carry information such as pre and post-conditions between a service user and a service. Dai et al. also present a framework that is capable of generating test cases and oracles, monitoring test execution and verifying the SUT. Bai et al. [6] propose a testing ontology model that describes test relations, concepts and semantics which can serve as a contract among test participants. Bai et al. [7] also propose a collaborative testing framework with contracts. The proposed approach is discussed in more detail in Section 10.

Contract-based testing of web services can achieve better and more efficient testing than WSDL-based testing due to the increased testability it provides. It also allows more efficient automated test case generation and test oracle creation. Regardless of the benefits that the contracts provide, the cost of creating contracts makes DbC less widely practiced. The same problem is also faced by semantic web services where specifications provide similar information to contracts.

9 Fault-Based Testing of Web Services

According to Morell [99], fault-based testing aims to prove that the SUT does not contain any prescribed faults. The difference between fault-based test cases and regular test cases is that the fault-based test cases prove the nonexistence of known faults rather than trying to find faults that exist.

Fault-based testing can test web services for common faults that can exist in the SOA environment and increase the dependability of services. Hanna and Munro [55] classify test data generation for different testing techniques and also surveyed the fault-based testing research in the web services domain. These testing techniques are:

1. *Interface propagation analysis* that is performed by randomly perturbing the input to a software component.
2. *Boundary value based robustness testing* where test data is chosen around the boundaries of the input parameter.
3. *Syntax testing* with invalid input where the rules of the specification of the input parameter are violated.
4. *Equivalent partitioning with invalid partition class* where the input space or domain is partitioned into a finite number of equivalent classes with invalid data

In this survey the categorization of the research undertaken in fault-based web service testing is done according to the level that faults are generated. Fault-based testing of web services can be grouped into three different categories according to the level that faults are applied to; XML/SOAP message perturbations, network level fault injection and mutation of web service specifications.

9.1 XML/SOAP perturbation

XML/SOAP perturbations are performed by capturing SOAP messages among services and their users. Faulty messages are generated from these captured messages by injecting faults before sending them or just by sending a faulty SOAP message to the web service. After perturbations, web service's behaviour with the faulty message is observed for verification.

One of the earliest examples of SOAP perturbation is proposed by Offutt and Xu [105]. Offutt and Xu propose three different types of perturbations;

1. *Data Value Perturbation* (DVP) that is performed by modifying the values in a SOAP message.
2. *Remote Procedure Calls Communication Perturbations* (RCP) that is performed by modifying the arguments of the remote procedures. Offutt and Xu propose the application of mutation analysis to syntactic objects and data perturbation to SQL code. SQL code perturbation also facilitates SQL injection testing.
3. *Data Communication Perturbations* (DCP) that is is used for testing messages that include database relationships and constraints

Xu et al. [179] propose an approach where perturbation is applied to XML Schemas in order to generate test cases. Xu et al. define XML Schema perturbation operators for creating invalid XML messages by inserting or deleting data fields. Almedia and Vergilio [36] also adopt the same approach and propose a tool called SMAT-WS that automates the testing process. Almedia and Vergilio also introduce some new perturbation operators for XML perturbation. Samer and Munro [55] test robustness of services by violating the input parameter specifications from WSDL files. Samer and Munro's approach can test both the web service itself and the platform the service resides in. Zhang and Zhang [186] propose a boundary value fault-injection testing in order to help with selecting the reliable web services. Similarly, Vieira et al. [158] propose a framework that performs fault-injection to the captured SOAP messages. Martin et al. [91] propose a framework called WebSob that tests web services for robustness. WebSob tests web service methods with extreme or special parameters.

Tsai et al. address the problems of test data ranking and fault-based testing within a single approach. Tsai et al. [155] propose an approach based on boolean expression analysis that can generate both true and false test cases. The proposed approach is supported by a framework that can rank test cases according to the test cases' likeliness to reveal errors.

The results of SOAP perturbations prove the effectiveness of this approach in revealing faults in web services. For example, during Offutt and Xu.'s experiments 18 faults are inserted into the Mars Robot Communication System (MRCS) and 100 DVP, 15 RCP and 27 DCP tests are generated. The generated tests achieved 78% fault detection rate in seeded faults (14 out of 18) and also revealed two natural faults. Xu et al. also experimented on MRCS and additionally on the supply chain management application from WS-I and achieved 33% fault detection. Almedia and Vergilio ran their experiments on a system consisting of nine web services and revealed 49 faults of which 18 of them are seeded by SMAT-WS. Vieira et al. experimented on 21 public web services and observed a large number of failures; however, 7 of these services showed no robustness problems. Vieira et al. also highlight that a significant number of the revealed errors are related to database accesses. Tsai et al. experimented on 60 BBS web services with 32 test cases and in these experiments negative test cases revealed more faults than positive test cases.

9.2 Network Level Fault Injection

Network Level Fault Injection (NLFI) is the fault-injection approach where faults are injected by corrupting, dropping and reordering the network packages. Looker et al. [87] propose the use of this technique along with a framework called Web Service Fault Injection Tool (WS-FIT). At the network level latency injection can be performed along with SOAP perturbation. WS-FIT can perform both SOAP perturbations and latency injections. Looker et al. also propose another fault-injection approach that simulates a faulty service. Faulty service injection is performed by replacing the values in SOAP messages with wrong values that are within the specified range of the parameter. Looker et al. also propose an extended fault model ontology that is used for generating faults and a failure modes ontology identifies the type of faults (seeded or natural fault).

Looker et al. experimented on a simulated stock market trading system that contains three web services. Baseline tests showed that latency injection caused the system to produce unexpected results 63% of the time. Faulty service injection results showed that the users do not encounter faults by the application of this injection method; however, unexpected results are noticed when the results of this method are compared with earlier tests.

9.3 Mutation of Web Service Specifications

Mutation testing is a technique used for measuring the adequacy of a test suite. It is also used in test data generation and test case management. Mutation testing is performed by injecting faults to SUT such as changing and modifying the source code or changing the SUT's state during test execution. These changes are applied using the rules of transformation called mutation operators.

In web services mutation operators are generally applied to the web service specifications. WSDL specifications allow creation of mutants at the interface level. The other levels of mutation such as source code or object level are not considered by the research community due to limitations in SOA. The mutation score at interface level is determined by the number of errors caused during the execution of test cases by invoking the faulty interfaces.

Naturally, one of the first examples of web service mutation testing is applied to WSDL specifications. Siblini and Mansour [134] propose the use of WSDL mutation for detecting interface errors in web services. For generating mutant web service specifications, nine mutation operators are defined for three different groups of WSDL elements:

1. *Types* element: There are seven different operators for this element.
   (a) Switch types in a complex data type.
   (b) Switch attributes of the same types in a complex data type.
   (c) Add or delete an element in a complex data type.
   (d) Add or delete an optional attribute in a complex data type.
   (e) Set the nil attribute to true in a complex data type.
   (f) Switch the elements of same data types.
   (g) Switch the attributes of same data types.
2. *Messages* element: switch parts between same message types.
3. *PortType* element: switch messages of same types in operation element.

Mei and Zhang [94] define mutation operators for contracts. The contracts in this approach are included in the extended WSDL specifications that are proposed by the authors. The mutated contracts are used in test data selection. The mutation operators defined by the Mei and Zhang are:

1. Contract negation operator.
2. Condition exchange operator.
3. Pre-condition weakening operator.
4. Post-condition strengthening operator.
5. Contract stuck-at operator.

The next step in web service mutation testing was to take the mutation into the semantic web services. The amount of information provided by OWL-S allow application of mutation operators at different levels compared to WSDL mutation. For example, Lee et al. [75] propose an ontology-based mutation to measure semantic test adequacy for composite web services and for semantic fault detection. Lee et al. define four types of mutations for OWL-S:

1. *Input/Output data mutation* where mutation operators modify the OWL ontology class definitions.
2. *Condition mutation* where mutation operators modify the specification of a condition.
3. *Control flow mutation* where mutation operators replace a control construct by a valid alternative.
4. *Data flow mutation* where mutation operators change the definitions and properties of different dependencies.

Similarly, Wang and Huang [160] propose another ontology-based mutation testing that uses OWL-S requirement model. Wang and Huang suggest a modified version of the requirement model enforced with Semantic Web Rule Language (SWRL) [142] in order to help with mutant generation. The proposed approach uses the enforced constraints in this model to generate mutants using Aspect Oriented Programming approach. The AOP allows evading the need of the original source code for mutant generation. Wang and Huang define eight different mutation operators:

1. *Empty parameter* that replaces the parameters of a method with empty string or null value.
2. *Exchange parameter* that swaps the parameters of a method.
3. *Insert unary operator* that applies operators to parameters.
4. *Parameter shift* that applies the shift operator to parameters.
5. *Parameter length transform* that modifies the constraints that define the lengths of data types.
6. *Parameter assignment* that sets the parameters to their boundary values using the defined constraints.
7. *Child service response time transform* that delays the invocation of other services if a timeout constraint is defined.
8. *Child service interface swap* that replaces the interfaces of the services that are invoked at invocation time.

According to the results of the proposed approaches, mutation testing is effective for measuring the test case adequacy in web services. Mei and Zhang's [94] WSDL mutation achieved 95%, Wang and Huang's [160] OWL-S mutation achieved 98.7% and Lee et al.'s [75] OWL-S mutation achieved 99.4% mutation score. Results also proved the effectiveness of mutation testing in test case selection. Mei and Zheng's approach achieved 96% and 81.5% reduction in test cases while maintaining the test coverage. Lee et al. approach also helped with equivalent mutant detection by detecting 25 equal mutants out of 686 generated mutants.

There is also another fault-injection approach by Fu et al. [49] that differs from the main stream at the level that the faults are injected. Fu et al. propose a fault-injection framework that is capable of performing white-box coverage testing of error codes in Java web services using compiler-directed fault injection. The fault injection is performed with the guidance of the compiler around try and catch blocks during run-time. Fu et al.'s approach achieved over 80% coverage on fault handling code during experiments.

Laranjeiro et al. [73] present a public web service robustness assessment tool called *wsrbench* [175]. Wsrbench provides an interface for sending SOAP messages with invalid web service call parameters. Wsrbench is also capable of providing the tester with detailed test results for a web service.

Fault-based testing of web services can be a very effective testing technique when the service user wants to check for common errors such as interface errors, semantic errors and errors that can be caused by the web service platform. Since the aim of fault-based testing is to observe the behaviour of a system with faulty data, using fault-based testing error handling code can also be tested and verified.

10 Collaborative Testing

Collaborative software testing is the testing concept where multiple parties involved in a web service, such as developer, integrator, tester and user, participate in testing process. Collaborative testing is generally used in testing techniques such as usability walk-throughs where correct functionality is tested with participation of different parties.

Challenges involving testing service-oriented systems is identified by Canfora and Di Penta [22] and some of these challenges require collaborative solutions. These challenges that might require collaborative solutions are:

1. Users not having a realistic test set.
2. Users not having an interface to test web service systems.
3. The need for a third-party testing and QoS verification rather than testing by each service user.

Tsai. et al. [148] propose a Co-operative Validation and Verification (CV&V) model that addresses these challenges instead of the traditional Independent Validation and Verification (IV&V). In this collaborative approach all parties of a web service such as UDDI broker, providers and clients also participate in the same testing process.

One example of this collaborative testing approach is Tsai et al.'s proposed enhanced UDDI server [151]. This UDDI server further enhances the verification enhancements in UDDI version 3 [157]. These proposed enhancements include:

1. The UDDI server stores test scripts for the registered web services.
2. The UDDI server arranges test scripts in a hierarchical tree of domains and sub-domains.
3. The UDDI server has an enhanced registration mechanism called *check-in*. Check-in mechanism registers a web service if it passes all test scripts for its related domain and sub-domain.

4. The UDDI server has a new mechanism before the client gets to use the selected service called *check-out*. This mechanism allows the client to test any web service before using it with the test scripts from web service's associated domain.

5. UDDI server includes a testing infrastructure that allows remote web service testing.

For the *check-in* process the SUT is tested using the existing scenarios in the SUT's domain. System scenarios for domain testing are expected to be obtained from common workflows for that domain which is defined by other organizations such as Web Services Interoperability (WS-I) organization [163]. The sub-domain scenarios are created by adding more constraints to the domain scenarios of the domain to which the web service belongs. The enhanced UDDI broker framework is also capable of generating test scripts for testing at different levels. The framework is capable of generating:

1. Method testing scripts that aim to test the methods in a web service.

2. Object testing scripts that aim to test a web service as an object ignoring other services that are involved in the application.

3. Integration testing scripts that aim to test a web service for a composite application with multiple services.

4. System integration testing scripts that aim to verify the whole system's functionality. System integration testing scripts can be generated from system requirements.

5. Domain testing scripts that aim to test all registered web services that belong to a certain domain or sub-domain.

Tsai et al. also suggest that a provider might want to provide test scenarios to be used at *check-out*. In the proposed framework, web service provider, as suggested by Canfora and Di Penta, can also provide test scripts to point out qualities of the web service such as robustness, performance, reliability and scalability. The proposed framework also provides an agent-based testing environment that automates the testing process both at *check-in* and *check-out*.

Tsai et al.'s proposed testing approach provides many benefits. such as increasing the quality of testing by providing realistic test cases from the service provider. This approach also reduces the number of test runs by testing web services before the binding process. The approach being automated also reduces the cost of testing. The framework's ability to generate scripts for different levels of testing makes it a complete testing solution. However, the framework's dependence on existing workflows might be a problem for some web services.

Bai et al. [7] also propose a contract-based collaborative testing approach that extends Tsai et al.'s enhanced UDDI proposal. Bai et al. propose a Decentralized Collaborative Validation and Verification (DCV&V) framework with contracts. The proposed framework consists of distributed test brokers that handle a specific part of the testing process. Bai et al. suggest two types of contracts for the DCV&V approach: Test Collaboration Contracts (TCC) that enforce the collaboration among the test brokers and Testing Service Contract (TSC) that is used for contract-based test case generation. The contract-based test case generation is explained in Section 8. Test broker architecture and collaboration among participants is explained in Figure 5.

The web service selection and testing process proceed collaborations, where:

1. The test provider registers the test artefacts to the test broker and the test information becomes available for testers.

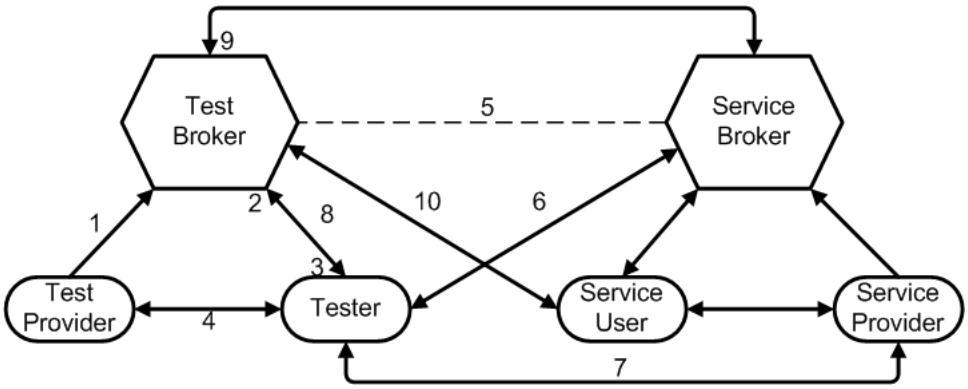

Figure 5. Bai et al.'s test broker architecture [7]

2. The tester looks for test cases initially rather than the web service itself.
3. The tester gets the test case description, test scripts or the link of the test service that the service provider selected.
4. This collaboration is needed in the case of the tester not being able to carry out the testing by himself.
5. The mapping between the test cases and the associated web services is established by the test broker.
6. Information on the web services that are associated with the selected test cases is delivered to the tester.
7. Web service at the service provider's location is tested with the test cases.
8. The test results are submitted to the test broker for further evaluation.
9. Collaboration between the test broker and the service broker can act like the Check-in and Check-out mechanism as in Tsai et al.'s enhanced UDDI broker proposal.
10. The service user is presented with the selected services test result in order to help with service selection. Service selection can be solely based on the test result in some cases.

Bai et al.'s proposed test broker provides a test case repository for test cases and test scripts, collects test results, maintains defect reports and web service evaluations. Test broker can generate and execute test cases as well. Bai et al. suggest that using DCV&V architecture multiple test brokers can involve in testing process. The decentralized architecture allows flexible and scalable collaborations among the participants.

Bai et al.'s contracts allow the test provider to provide specification-based test case designs for the other participants. Testers can also run synchronized tests on services and publish the test results. Test data and test knowledge can be made accessible to others and can be exchanged among different parties. Generally, contracts aim to enforce the correct functionality of a system but Bai et al.'s contracts take it to a different level. Bai et al.'s contracts enforce collaboration among parties. Enforcing the collaboration in SOA using contracts can be an effective approach but the cost of generating contracts might be discouraging.

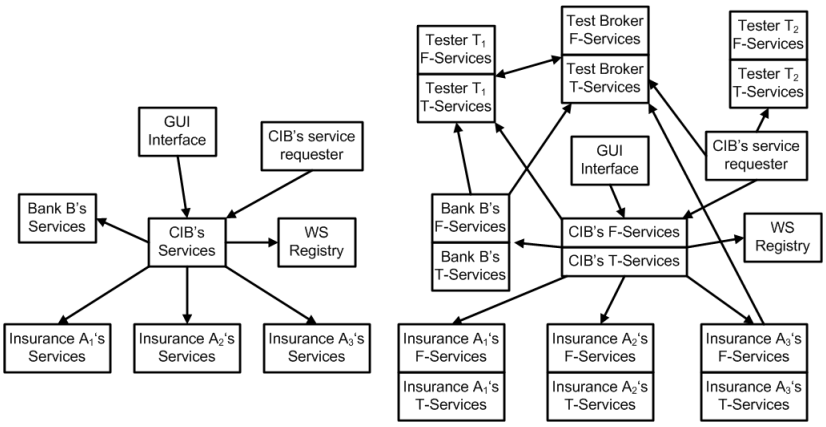

Figure 6. Zhu's proposed Service-Oriented Testing Environment [191]

Zhu [191, 187] proposes another collaborative approach. In Zhu's approach service developers or third party testers provide testing services that help with testing the actual services. Zhu also proposes a testing ontology that is based on taxonomy of testing concepts called the STOWS. The proposed ontology aims to solve the issues related to the automation of test services.

Figure 6 shows the overview of a Car Insurance Broker (CIB) service and the environment proposed by Zhu for the same CIB service. The proposed environment consists of the actual services and their testing agents. Services starting with "T" represent a testing service and services starting with "F" represent a functional service. Testing services enable the tester to execute tests on the associated F-service through two types of mechanisms; observation mechanisms and control mechanisms.

Observation mechanisms allow the service user to observe the system behaviour for unexpected occurrences such as unexpected usages of the system, different kinds of system/message failures on different levels of the SOA environment and failures that can happen due to the errors in the provided specifications.

Control mechanisms allow the tester to control test execution on the F-services. F-services are the exact or similar copy of the original web service. The reason for providing such a service is to enable testers to have more control of test execution. The test executions that use the original web service might cause problems, such as having test data in the developer system or service disruptions due to high test loads. F-services not only solve these issues but also provide more flexible test executions for the tester.

The main advantage of Zhu's proposed testing environment is that testing can be fully automated. Another advantage is that the SUT is not affected by the testing process: there will be no service disruptions due to any error that might happen during testing process or a decrease in the service's performance. Proposed environment also reduces the security concerns by allowing tests to be performed via testing servers. The only disadvantage of the proposed environment is that the testing services need to be tested as well. This problem can be solved by the use of certified third party testing services which require no testing. Using third party testing services might increase the cost vastly due to the costs of testing services.

Bartolini et al. [9] introduce a collaborative testing approach that "whitens" the web service testing by introducing a new stakeholder called TCov that provides the tester with coverage information. Bertolini et al.'s approach requires the service provider to insert instrumentation code inside the service in order to provide TCov with coverage information. This provided information is then analysed by TCov provider

and is made available to the tester as a service.

Proposed collaborative testing approaches aim to solve some of the challenges involved in web service testing. For example, having an adequate test suit by allowing service providers to provide test suits or having a third party tester providing testing interfaces for service users. The claimed benefits of these approaches prove the necessity of collaborative testing in service-oriented environments.

11 Regression Testing of Web Services

Regression testing is the reuse of the existing test cases from the previous system tests. Regression testing is performed when additions or modifications are made to an existing system. In traditional regression testing it is assumed that the tester has access to the source code and the regression testing is done in a white-box manner [**?**]. Performing white-box regression testing helps mainly with test case management.

A number of the common test case management and prioritization methods such as the symbolic execution approach and the dynamic slicing-based approach require testers' access to the source code [**?**]. However, approaches like the Graph Walk Approach (GWA) by Rothermel and Harrold [125] that does not require access to the source code, can be used in Web Services Regression Testing (WSRT). Later in this section the application of this approach will be elaborated. The important reason for distinguishing the Regression Test Selection (RTS) methods that require access to source code is to point out the applicable RTS methods for testing web services. Since web service testing is considered a black-box testing, RTS methods that require access to the source code are inapplicable to web services.

According to Canfora and Di Penta [22], one of the main issues in WSRT at the user side is not knowing when to perform regression testing. Since the service user has no control over the evolution of the web service, the service user might not be aware of the changes to the web service. There are two possible scenarios for informing the service user about the modifications. These scenarios are based on the service provider's knowledge about the service users.

The first scenario arises when the SUT is registered to a UDDI broker or is not a free service. The subscription service in UDDI v3 allows automated notification of the users when changes are made to a service. For paid services it is assumed that the service provider has the details of the service users through billing means or service agreements. In this scenario informing the service users about the changes that are made to a web service will not be a problem. Even so, there is still a small room for error. If service users are not properly informed about which methods of the web service are modified, they might either perform the series of tests even though the functions that they use are not affected or fail to perform the necessary tests due to lack of information.

The second scenario arises when the web service that requires regression testing is a public web service with no UDDI registration and the provider does not have information about its users. This scenario is the most problematic one, because the service user can only be aware of the modifications by observing errors in system behaviour or changes in the system performance. During the period between changes being made to a service and its user discovering the changes, the confidence in the service might decrease due to errors or decreases in quality of service.

Another challenge in WRST is the concurrency issues that might arise during testing due to the tester not having control over all participating web services. Ruth and Tu [127] discussed these issues and identified possible scenarios. Even though they identified three different scenarios, all of them are based on the issue of having a service or a method modified other than the SUT during the regression test process. The problem attached to this issue is called *fault localization*. During the regression testing process, if a tester is not informed about the modifications to a web service that is invoked by the SUT, then the faults that are caused by this service can be seen as the faults in the SUT.

As explained earlier, the RTS method of Rothermel and Harrold is used by many researchers of WSRT. The proposed approaches by the researchers usually differ in the method of the Control Flow Graphs (CFG) generation. CFG is a representation language that uses graph notation for representing all the paths that might be traversed during a software's execution [30].

Ruth and Tu [128] propose a regression testing approach that is based on Rothermel and Harrold's GWA technique. This approach assumes that the CFGs of participating services are provided by their developers. Ruth and Tu also propose that the test cases and a table of test cases' coverage information over the CFG must also be provided along with WSDL file via WS-Metadata Exchange Framework [171]. The required CFG needed to be at statement level, meaning every node in the CFG will be a statement. These nodes will also keep a hash code of their corresponding statements. When a change is made to the system, the hash of the modified service will be different from the hash of the original service so that the RTS algorithm detects the modified parts in the service without seeing the actual source code.

By gathering the CFG from the invoked web services, a global CFG can be constructed in order to apply GWA. Every time a change is made to a service a new CFG is formed. The user becomes aware of any changes to the web service by comparing the new CFG with the old one using dual-traversal comparison.

Ruth et al. [126] also propose an automated extension to their RTS technique that tackles the concurrency issues that might arise during WSRT. This approach helps in solving the multiple modified service problem by using *call graphs* [128]. It is possible to determine the execution order of the modified services by using the call graphs. A strategy called "downstream services first" is applied in order to achieve fault localization. In this strategy if a fault is found in a downstream service, none of the upstream services are tested until the fault is fixed. Ruth et al. also take the situation into consideration where a service makes multiple calls to different services in parallel.

Ruth et al.'s approach also provides a solution for automating regression testing. There are two proposed ways of automating regression testing. The first way is using the call graphs and a central service that constructs the global CFG. When a service is modified, the host of the service informs the central service and the regression testing process is performed as explained earlier. The second way is using the publisher/subscriber method [120] where a list of all service users are kept in a log. Ruth and Tu's also present a framework that provides this functionality.

Ruth et al.'s regression testing approach can be very efficient if all CFGs for called services are available and granularities of CFGs are matched but it also requires the developers to create CFGs and hashes. This approach is also limited to static composite services due to the complexity of generating CFGs for composite services.

Lin et al. [83] propose another GWA based regression testing approach where CFGs are created from Java Interclass Graph (JIG) [56]. A framework that performs RTS on the transformed code of a Java based web service is also proposed by Lin et al. The code transformation can be performed only in Apache Axis framework [2]. The proposed approach uses the built-in WSDL2Java [174] generated classes both on the server and user side and replaces messaging with local method invocation. A simulation environment is created by combining stub and skeleton objects into a local proxy in a local Java virtual machine. Execution of the simulation allows the generation of JIG on which the GWA can be performed. Compared to the previously presented approaches, Lin et al.'s approach is able to generate the CFG in an automated fashion without the knowledge of internal behaviour of the web service. The main limitation of this approach is its applicability being limited to Java-based web services.

Liu et al. [85] address the issues that occur due to concurrency in BPEL regression testing. Lui et al. propose a test case selection technique based on impact analysis. The impact analysis is performed by identifying the changes to the process under test and discovering impacted paths by these changes.

Mei et al. [96] propose a different black-box test case prioritization technique for testing web service

compositions. In this approach test case prioritization is based on the coverage of WSDL tags in XML schemas for input and output message types.

Tarhini et al. [144] propose another model-based regression testing approach using the previously explained model in Section 7. The proposed model is capable of representing three types of modifications to the composite services:

1. Addition of a new service to the system.
2. Functional modifications to an existing service in the system.
3. Modifications to the specification of the system.

The changes that are made to the system are represented in the modified version of the original TLTS. The second and the third type of modifications are represented by adding or removing states or edges from the TLTS of the original system.

TLTS is capable of representing the changes that are made to the system both at the service's internal level and at the application level. This capability enables the selection algorithm to select the necessary test cases for each type of modification. Tarhini et al.'s model-based testing approach is also capable of generating test cases for the new and existing services and operations. The ability to perform test history updates at the end of each run of the test selection algorithm also accomplishes test case management.

An approach that performs regression testing for BPEL processes is proposed by Wang et al. [159]. This approach uses the BFG [185] that is explained in Section 4.2. Wang et al. propose a BPEL regression testing framework that can generate and select test cases for regression testing using Rothermel and Harrold's RTS technique. Wang et al. extended the BFG model into another graph called eXtensible BFG (XBFG) that is claimed to be more suitable for regression testing.

The need for a visual testing tool is addressed by Pautasso [114]. The author proposes a framework called JOpera. JOpera is capable of performing unit and regression testing on web service compositions. The proposed tool is implemented using a language called JOpera Visual Composition Language [115].

Tsai et al. [156] propose a Model-based Adaptive Test (MAT) case selection approach that can be applied to both regression testing and group testing. This approach defines a model called Coverage Relationship Model (CRM) that is used for test case ranking and selection. Using this model, test cases with similar aspects and coverage can be eliminated. Tsai et al. define multiple rules that guarantee the selection of the most potent test cases and prove that the less potent test cases never cover the more potent test cases. Tsai et al. claim that this approach can be applied to regression testing when a new version of a service with the same specifications is created.

In WSRT, test case management has other test case prioritization considerations such as service access quotas. Hou et al. [60] address the issue of quota-constraints for WSRT. The quota problem might occur when performing regression testing on web services with a limited number of periodic accesses. Having quotas can affect a service user in two ways:

1. It might increase the cost of testing if the service user is on a pay per use agreement. Each time a test case is executed, the cost of testing will increase.
2. It might cause an incomplete test run if the service user runs out of access quota before completing the regression test.

Hou et al. propose a scheduled regression testing that divides the testing process into several parts according to the user's access quotas while ignoring the actual execution time of the regression testing. The aim of this approach is to divide test cases into groups based on time slots that suit the user's web service access

quotas. Proposed test case prioritization approach is based on a multi-objective selection technique which defines an objective function that aims to attain maximum coverage within the quota constraints of services.

Di Penta et al. [37] propose a collaborative regression testing approach that aims to test both functional and non-functional properties of web services. Di Penta et al.'s approach allows the developer to publish test cases along with services that are used in the initial testing and regression testing. The approach also reduces the cost of regression testing by monitoring service input-output. All these functionalities are provided by a testing tool that supports this approach.

WSRT presents challenges that do not exist in testing traditional software, due to the nature of web services. The issues that were pointed out in WSRT by Canfora and Di Penta [22], at both the user and the provider side, are addressed by some of the work discussed in this section. For example, the issue of users' not having a realistic test suite is addressed by Ruth and Tu [128] and the concurrency issues are addressed by Ruth and Tu [126]. Other additional concerns in WSRT such as quota issues are also addressed [60]. The main problem of automating the process of informing service users about the changes to a service is also addressed by Ruth and Tu [128].

12 Interoperability Testing of Web Services

Interoperability is the ability of multiple components to work together, that is, to exchange information and to process the exchanged information. Interoperability is a very important issue in open platforms such as SOA. Even though web services must conform to standard protocols and service specifications, incompatibility issues might still arise.

The need for interoperability among service specifications is recognized by the industry and WS-I, an open industry organization, formed by the leading IT companies. The organization defined a WS-I Basic Profile [162] in order to enforce web service interoperability. WS-I organization provides interoperability scenarios that need to be tested and a number of tools to help testing process. Kumar et al. [132] describe the possible interoperability issues regarding core web service specifications such as SOAP, WSDL and UDDI and explain how the WS-I Basic Profile provides solutions for the interoperability issues with web service specifications.

The need for testing the interoperability among services is also recognized by researchers. For example, Bertolino and Polini [15] propose a framework that tests the service interoperability using service's WSDL file along with a Protocol State Machine Diagram (PSM) [118] provided by the service provider. The PSM diagram carries information on the order of the operation invocations for a service. The proposed framework checks the order of the web service invocations among different web services and tries to point out possible interoperability problems.

Yu et al. [184] propose an ontology-based interoperability testing using the communication data among web services. The proposed approach captures the communication data and stores it in an ontology library. The data in the library is analysed and reasoning rules for error analysis and communication data is generated in order to run with JESS reasoning framework [66]. Using the rules from JESS framework gives this approach the ability to adapt certain problems such as network failure or network delay. The framework can also replay the errors that have been identified by using a Petri-Net graphic of the communicating web services.

Smythe [136] discusses the benefits of the model-driven approach in SOA context and proposes an approach that can verify interoperability of services using UML models. The author points out the need for UML-profile for SOA that contains the interoperability specification in order to use with the proposed approach. Using the proposed the UML-profile, test cases can be generated for testing the conformance of the web service's interoperability.

Similarly, Bertolino et al. [15] propose another model-based approach for testing interoperability. The proposed framework is similar to the Tsai et al.'s extended UDDI broker [151] where web services are tested before registration. In this approach, service provider is expected to provide information on how to invoke the web service using an UML 2.0 behaviour diagram.

Ramsokul and Sowmya [122] propose a framework that verifies the conformance of web service protocol's implementation to the protocol's specification. Ramsokul and Sowmya propose the use of the ASEHA (referred to in Section 7). Ramsokul and Sowmya claim that the ASEHA framework is capable of modelling complex protocols such as Web Services Atomic Transaction (WS-AtomicTransaction) [167] and Web Services Business Activity (WS-BusinessActivity) [168]. The proposed ASEHA framework captures the SOAP messages from services, maps them into ASEHA automata and verifies the protocol implementation against its specification.

Betin-Can and Bultan [16] propose the use of a model based on HSMs for specifying behavioural interfaces of participating services in a composite service. Betin-Can and Bultan suggest that verification of the web services that are developed using Peer Controller Pattern is easier to automate and propose the use of HSMs as the interface identifiers for web services in order to achieve interoperability. Betin-Can and Bultan propose a modular verification approach using Java PathFinder to perform interface verification and SPIN for behaviour verification and synchronizability analysis. The use of the proposed approach improves the efficiency of the interface verification significantly as claimed by the Betin-Can and Bultan.

Narita et al. [102] propose a framework for interoperability testing to verify web service protocols especially aimed at reliable messaging protocols. Narita et al. claim that there is no test coverage in communication protocols and there is a great need for execution of erroneous test cases for these protocols. Test cases containing erroneous messages are created by the framework by intercepting messaging across web services and observing error recovery of the service.

As stated, interoperability is one of major advantages of web services. Web services must be tested for interoperability in order to provide the level of interoperability SOA requires. Interoperability issues that are caused by different versions of the protocols such as SOAP are addressed by industry and other projects such as WS-I and ILab. However, the rest of the challenges requires approaches such as Tsai et al. [151] and Bertolino et al.'s [15] where interoperability is tested before service registration. Testing web services before the registration process can prevent many of the possible interoperability problems and this might increase the confidence in the registered services.

13 Integration Testing of Web Services

It is important to perform integration testing in software engineering to make sure all the components work as a system. Since the idea behind SOA is to have multiple loosely coupled and interoperable services to form a software system, integration testing in SOA becomes a necessity. By performing integration testing, all the elements of SOA can be tested including services, messages, interfaces, and business processes.

Bendetto [12] defined the difference between integration testing of traditional systems and service-oriented systems. Canfora and Di Penta [22] point out the challenges in integration testing of SOA. According to Bendetto and Canfora and Di Penta the challenges of integration testing in service-oriented environments are:

1. Integration testing must include the testing of web services at binding phase, workflows and business process connectivity. Business process testing must also include all possible bindings.

2. Low visibility, limited control and the stateless nature of SOA environment make integration testing harder.

3. Availability of services during testing might also be a problem.

4. Dynamic binding makes the testing expensive due to the number of required service calls.

One of the earliest work in integration testing of web services is Tsai et al.'s [152] Coyote framework. Coyote is an XML-based object-oriented testing framework that can perform integration testing. Coyote is formed of two main components; a test master that is capable of mapping WSDL specifications into test scenarios, generating test cases for these scenarios and performing dependency analysis, completeness and consistency checking and a test engine that performs the tests and logs the results for these tests. Coyote performs WSDL-based test data generation.

In software development there is a concept called Continuous Integration (CI) [41]. CI is performed by integrating the service under development frequently. CI also requires continuous testing. Continuous Integration Testing (CIT) allows early detection of problems at the integration level. Huang et al. [63] propose a simulation framework that addresses the service availability problem using CIT. The proposed framework automates the testing by using a surrogate generator that generates platform specific code skeleton from service specifications and a surrogate engine that simulates the component behaviour according to skeleton code. Huang et al. claim that the proposed surrogates are more flexible than the common simulation methods such as stubs and mocks and the simulation is platform-independent.

Peyton et al. [117] propose a testing framework that can perform "grey-box" integration testing on of composite applications and their underlying services. The proposed framework is implemented in TTCN-3 [45], an ETSI standard test specification and implementation language, and is capable of testing the composite application behaviour and interaction between participating web services. The framework increases the visibility and the control in testing by inserting test agents into a web service composition. These agents are used in analysing HTTP and SOAP messages between the participating services.

Mei et al. [95] address the integration issues that might be caused by XPath in BPEL processes such as extracting wrong data from an XML message. The proposed approach uses CFGs of BPEL processes along with another graph called XPath Rewriting Graph (XRG) that models XPath conceptually (models how XPath can be rewritten). Mei et al. create a model that combines these two graphs called X-WSBPEL. Data-flow testing criteria based on def-use associations in XRG are defined by Mei et al. and using these criteria, data-flow testing can be performed on the X-WSBPEL model.

Tarhini et al.[143] address the issue of web service availability and the cost of testing. Tarhini et al. solve these problems by finding the suitable services before the integration process and using only the previously selected services according to their availability. In this approach, testing in order to find suitable services is accomplished in four stages. First stage is the "find stage" where candidate web services from a service broker is found. In the second stage selected web services are tested for their correct functionality. At the third stage each web service is tested for interaction as a stand-alone component and if it passes this stage, it is tested for interaction with the rest of the components. When a web service passes all the required steps it is logged into the list of services to be invoked at runtime. The proposed framework uses a modified version of the Coyote framework [152] for the automation of testing.

Yu et al. [183] address the interaction problems within OWL-S compositions. Yu et al.'s approach tests interaction among participating web services using interaction requirements. Yu et al. propose an extension to existing OWL-S model in order to carry these requirements.

There are also previously mentioned approaches that are capable of performing integration testing. For example, Tsai et al.'s ASTRAR framework [147] and the proposed Enhanced UDDI server [151] that are discussed in Section 6 and 10 are also capable of performing integration testing. Similarly Lenz et al.'s [77] model-driven testing approach can be used for integration testing.

Integration testing is one of the most important testing techniques in SOA assuring the correct functioning of service-oriented systems. The challenges that a tester faces during integration testing are addressed by some of the approaches mentioned in this section such as Tarhini et al. [143] and Huang et al.'s [63] frameworks. Effective integration testing requires effective scenarios that capture all functionality of the SUT and all possible scenarios of execution.

14 Conclusion

SOC changed the business understanding of the whole software industry. However, the change from traditional software to services and the service usage is still not at the expected rate. One of the most important issues that prevent the wider use of web services is the issue of trust. One of the effective solutions to this trust issue is testing.

This survey has focused on the functional testing techniques and approaches that have been proposed for testing web services. Testing web services is more challenging than testing traditional software due to the complexity of web service technologies and the limitations that are caused by the SOA environment. The complexity of web services due to their standards not only affects the testing but also slows down the transition to web services. Limited control and ability to observe render most of the existing software testing approaches inapplicable.

There are other issues in web service testing, such as:

1. The frequency of testing required,
2. Testing without disrupting the operation of the service,
3. Determining when testing is required and which operations need to be tested,

that do not exist for traditional software systems. As web services capture more attention from both the industry and the research communities, more issues involving testing web services are being identified. Some of the previously identified issues are addressed by the approaches discussed in this survey and others still need attention.

This survey covered unit testing, regression testing, integration testing and interoperability testing of web services. These testing techniques test important functional aspects of web services. Testing methods such as model-based testing, fault-based testing, formal verification, partition testing, contract-based testing and test case generation are also surveyed. Some of the surveyed testing approaches where multiple parties of a web service participate in testing are categorized under collaborative testing. Other existing testing methods that aim at testing the non-functional aspects of web services, such as security or QoS, are outside the scope of this survey.

Acknowledgments

Lorna Anderson and Kathy Harman assisted with proof reading. The authors sent this paper to those cited seeking comments and were very grateful for the many authors who responded, often in considerable detail. Overall, responses were received from 40 authors, making it sadly infeasible to acknowledge all of them here by name.

Table 1: Testing Approaches

Author	Testing Approach	WS Technology	Tools	Experimented Web Services
Bartolini et al. [10]	Specification based test data generation	WSDL	WS-TAXI	Pico service
Ma et al. [89]	Specification based test data generation	WSDL	–	Synthetic order system (3 versions)
Bai et al. [5]	Specification based test data generation	WSDL	–	356 public services
Li et al. [81]	Specification based test data generation	WSDL	WSTD-Gen	Call center query service
Paradkar et al. [113]	Model-based testing	OWL-S	–	An online application for asset management
Dong and Yu [39]	Model-based testing	WSDL	–	–
Ma et al. [90]	Model-based testing	BPEL	–	–
Guangquan et al. [54]	Model-based testing	BPEL	–	Book renewal process
Conroy et al. [29]	Test data generation	WSDL	–	–
Yan et al. [180]	Model-based testing	BPEL	–	Loan approval
Yuan et al. [185]	Model-based testing	BPEL	–	–
Endo et al. [43]	Model-based testing	BPEL	ValiBPEL	GCD, Loan approval and nice journey
Kaschner and Lohmann [69]	Model-based testing	BPEL	–	Example online shop
Hou et al. [59]	Test case generation	BPEL	–	Loan approval(1-2), atm, market place, gymlocker, BPEL(1-5)
Blanco et al. [17]	Test case generation using search-based methods	BPEL	–	Loan approval
Bertolino et al. [14]	Partition testing	WSDL	TAXI	–
Bai et al. [6]	Partition testing	OWL-S	–	Synthetic travel system
Sneed and Huang [137]	Unit testing	WSDL	WSDLTest	eGoverment project with 9 Web Services (WSs)
Lenz et al. [77]	Model-driven unit testing	–	–	–
Chan et al. [25]	Unit testing using metamorhpic relations	–	–	–
Tsai et al. [147]	Unit testing	WSDL	ASTRAR	–
Mayer and Lübke [92]	Unit testing	BPEL	BPELUnit	–
Li et al. [79]	Unit testing	BPEL	–	–
Sinha and Paradkar [135]	Model-based testing	WSDL-S	–	–
Fu et al. [50]	Model-checking	BPEL	SPIN	–
García-Fanjul et al. [51]	Model-based testing	BPEL	SPIN	Loan approval example
Zheng et al. [189]	Model-based testing	BPEL	SPIN, NuSMV	–
Huang et al. [62]	Model-based testing	OWL-S	BLAST	Synthetic shopping WS
Betin-Can and Bultan [16]	Model-checking	BPEL	JavaPathfinder, SPIN	Synthetic travel agency service
Ramsokul and Sowmya [121]	Model-checking	WS protocols	SPIN	WS-AT
Lallai et al. [72]	Model-based testing	BPEL	BPEL2IF, TestGen-IF	Oracle example loan Service
Dong and Yu [39]	Model-based testing	WSDL	–	–
Wang et al. [161]	Model-based testing	OWL-S	TCGen4WS	–

Continued on Next Page. . .

Table 1 – Continued

Author	Testing Approach	WS Technology	Tools	Experimented Web Services
Ouyang et al [107]	Formal verifiction using Petri-Nets	BPEL	BPEL2PNML, WofBPEL	–
Schlingloff et al. [129]	Model-checking using Petri-Nets	BPEL	Lola	–
Lohmann et al. [86]	Formal verifiction using Petri-Nets	BPEL	BPEL2oWFN, Fiona	Onlineshop example
Yang et al.'s [181]	Formal verifiction using Petri-Nets	BPEL, WSCI	CPNTools	–
Yi et al. [182]	Formal verifiction using Petri-Nets	BPEL	CPNTools	Airline and travel agency service
Dong et al. [40]	Formal verifiction using Petri-Nets	BPEL	Pose++	–
Dai et al. [34]	Formal verifiction using Petri-Nets	BPEL	MCT4WS	–
Xu et al. [178]	Formal verifiction using Petri-Nets	BPEL	BPEL2PNML	–
Li et al. [78]	Model-based testing	WSDL	WS-StarGaze	Parlay X Conference WS, CSTA Routeing Service
Tarhini et al. [143]	Model-based testing	WSDL	–	–
Heckel and Lochmann [57]	Contract-based testing	WSDL	–	–
Mei and Zhang [94]	Contract-based testing	WSDL	–	2 synthetic services (Tritype and Middle)
Dai et al. [33]	Contract-based testing	OWL-S	CBT4WS	–
Offutt and Xu [105]	Fault-based testing	WSDL	–	Mars Robot Communication Service (MRCS)
Xu et al. [179]	Fault-based testing	WSDL	–	MRSC, WS-I example supply chain application
Almedia and Vergilio [36]	Fault-based testing	WSDL	SMAT-WS	9 goverment services
Samer and Munro [55]	Fault-based testing	WSDL	–	–
Vieira et al. [158]	Fault-based testing	SOAP	–	21 public WS
Tsai et al. [155]	Fault-based testing	OWL-S	–	60 BBS web services
Martin et al. [91]	Fault-based testing	WSDL	WebSob	–
Looker et al. [87]	Fault-based testing	SOAP	WS-FIT	Synthetic stock market trading system with 3 WS
Siblini and Mansour [134]	Fault-based testing	WSDL	–	Credit card checking service
Mei and Zhang [94]	Fault-based testing	WSDL	–	2 synthetic services (Tritype and Middle)
Lee et al. [75]	Fault-based testing	OWL-S	–	Book finder example OWL-S
Wang and Huang [160]	Fault-based testing	OWL-S	–	Synthetic banking system
Fu et al. [49]	Fault-based testing	–	–	4 Java web service applications (FTPD, JNFS, Haboob, Muffin)
Tsai et al. [151]	Collaborative testing	UDDI	–	–
Bai et al. [7]	Collaborative testing	–	–	–
Zhu [191]	Collaborative testing	–	–	–
Bartolini et al. [9]	Collaborative testing	–	–	–

Continued on Next Page. . .

Table 1 – Continued

Author	Testing Approach	WS Technology	Tools	Experimented Web Services
Ruth and Tu [128]	Regression testing	–	–	–
Ruth et al. [126]	Regression testing	–	–	–
Lin et al. [83]	Regression testing	–	–	Purchase order WS
Liu et al. [85]	Regression testing	BPEL	–	–
Mei et al. [96]	Regression testing	BPEL	–	6 BPEL examples from IBM repositiory (atm, buybook, gym-locker, loanapproval, market-place, purchase, triphandling) and buybook and dslservice
Tarhini et al. [144]	Regression testing	–	–	–
Wang et al. [159]	Regression testing	BPEL	–	Example online shop from Tools4BPEL
Tsai et al. [156]	Test case selection	WSDL, OWL-S	–	–
Pautasso [114]	Regression testing	–	JOpera	–
Di Penta et al. [37]	Regression testing	–	–	dnsjava converted to WS
Hou et al. [60]	Regression testing	–	–	–
Bertolino and Polini [15]	Interoperability testing	WSDL	–	–
Yu et al. [184]	Interoperability testing	OWL-S	–	–
Ramsokul and Sowmya [122]	Model-checking for interoperability	WS protocols	–	WS-AT
Smythe [136]	Interoperability testing	–	–	–
Betin-Can and Bultan [16]	Model-checking for interoperability	–	–	–
Narita et al. [102]	Interoperability testing	WS protocols	WS -VS	WS-Reliability
Tsai et al.'s [152]	Integration testing	WSDL	Coyote	–
Huang et al. [63]	Integration testing	–	–	HR management system
Peyton et al. [117]	Integration testing	SOAP,WSDL	TTCN-3	–
Mei et al. [95]	Integration testing	BPEL	–	8 open-source BPEL applications
Yu et al. [183]	Interaction testing	OWL-S	–	–

References

[1] S. Ali, L. C. Briand, H. Hemmati, and K. R. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test-case generation," *IEEE Transactions on Software Engineering*. Accepted for future publication.

[2] Apache Web Services Project - Apache Axis, [Online] http://ws.apache.org/axis/.

[3] AppLabs, "Web services testing a primer." [Online], http://www.docstoc.com/docs/4248976/Web-Services-Testing-A-Primer ,May 2007.

[4] C. Atkinson, D. Brenner, G. Falcone, and M. Juhasz, "Specifying high-assurance services," *Computer*, vol. 41, pp. 64–71, Aug. 2008.

[5] X. Bai, W. Dong, W. T. Tsai, and Y. Chen, "WSDL-based automatic test case generation for web services testing," in *SOSE 2005: Proceedings of the IEEE International Workshop on Service-Oriented System Engineering*, pp. 207–212, Beijing, China, Oct. 2005, IEEE Computer Society.

[6] X. Bai, S. Lee, W. T. Tsai, and Y. Chen, "Ontology-based test modeling and partition testing of web services," in *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*, pp. 465–472, Beijing, China, Sept. 2008, IEEE Computer Society.

[7] X. Bai, Y. Wang, G. Dai, W. T. Tsai, and Y. Chen, "A framework for contract-based collaborative verification and validation of web services," in *CBSE 2007: Proceeedings of the 10th International Symposium on Component-Based Software Engineering* (H. W. Schmidt, I. Crnkovic, G. T. Heineman, and J. A. Stafford, eds.), vol. 4608 of *Lecture Notes in Computer Science*, pp. 258–273, Medford, Massachusetts, USA, 2007, Springer.

[8] B. Banieqbal, H. Barringer, and A. Pnueli, eds., *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings*, vol. 398 of *Lecture Notes in Computer Science*, Springer, 1989.

[9] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti, "Whitening SOA testing," in *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 161–170, Amsterdam, The Netherlands, August 2009, ACM.

[10] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "WS-TAXI: A WSDL-based testing tool for web services," in *ICST '09: Proceedings of the International Conference on Software Testing Verification and Validation*, pp. 326–335, Denver, Colorado, USA, 2009, IEEE Computer Society.

[11] G. Bechara, "What is a reusable service?." [Online], http://www.oracle.com/technetwork/articles/bechara-reusable-service-087796.html ,March 2009.

[12] C. Benedetto, "SOA and integration testing: The end-to-end view." [Online], http://soa.sys-con.com/node/275057 ,September 2006.

[13] L. Bentakouk, P. Poizat, and F. Zaïdi, "A formal framework for service orchestration testing based on symbolic transition systems," in *TestCom/FATES: 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4, 2009. Proceedings*, vol. 5826/2009 of *Lecture Notes in Computer Science*, pp. 16–32, Springer, Eindhoven, The Netherlands, November 2009.

[14] A. Bertolino, J. Gao, E. Marchetti, and A. Polini, "Automatic test data generation for XML Schema-based partition testing," in *AST '07: Proceedings of the 2nd International Workshop on Automation of Software Test*, pp. 4–4, Minneapolis, Minnesota, USA, May 2007, IEEE Computer Society.

[15] A. Bertolino and A. Polini, "The audition framework for testing web services interoperability," in *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 134–142, Porto, Portugal, Aug. 2005, IEEE Computer Society.

[16] A. Betin-Can and T. Bultan, "Verifiable web services with hierarchical interfaces," in *ICWS '05: Proceedings of the 2005 IEEE International Conference on Web Services*, pp. 85–94 vol.1, Orlando, Florida, USA, July 2005, IEEE Computer Society.

[17] R. Blanco, J. García-Fanjul, and J. Tuya, "A first approach to test case generation for BPEL compositions of web services using scatter search," in *ICSTW '09: Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, pp. 131–140, Denver, CO, USA, 2009, IEEE Computer Society.

[18] BLAST, [Online] http://mtc.epfl.ch/software-tools/blast/.

[19] J. Bloomberg, "Web services testing: Beyond SOAP." [Online], http://searchsoa.techtarget.com/news/article/0,289 142,sid26_gci846941,00.html ,September 2002.

[20] D. Brenner, C. Atkinson, O. Hummel, and D. Stoll, "Strategies for the run-time testing of third party web services," in *SOCA 2007: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, vol. 0, pp. 114–121, Newport Beach, California, USA, June 2007, IEEE Computer Society.

[21] G. Canfora and M. Di Penta, "SOA: Testing and self-cheking," in *Proceedings of the International Workshop on Web Services Modeling and Testing (WS-MaTe2006)* (A. Bertolino and A. Polin, eds.), pp. 3–12, Palermo, Italy, June 2006.

[22] G. Canfora and M. Di Penta, "Testing services and service-centric systems: challenges and opportunities," *IT Professional*, vol. 8, pp. 10–17, March 2006.

[23] G. Canfora and M. Di Penta, "Service-oriented architectures testing: A survey," in *Proceedings of the 31st International Spring Seminar on Electronics Technology (ISSSE 2008)*, pp. 78–105, Budapest, Hungary, 2008.

[24] CBDI Forum, [Online] http://www.cbdiforum.com/.

[25] W. Chan, S. Cheung, and K. Leung, "Towards a metamorphic testing methodology for service-oriented software applications," in *Proceedings of the 5th International Conference on Quality Software (QSIC 2005)*, pp. 470–476, Melbourne, Australia, Sept. 2005, IEEE Computer Society.

[26] H. Chen, H. Jin, F. Mao, and H. Wu, *Web Technologies Research and Development - APWeb 2005*, ch. Q-GSM: QoS Oriented Grid Service Management, pp. 1041–1044. SpringerLink, 2005.

[27] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Tech. Rep. HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[28] E. M. Clarke and B.-H. Schlingloff, "Model checking," in *Handbook of Automated Reasoning* (J. A. Robinson and A. Voronkov, eds.), vol. 2, ch. 24, pp. 1635–1790, Amsterdam, The Netherlands: Elsevier Science Publishers B. V., 2001.

[29] K. Conroy, M. Grechanik, M. Hellige, E. Liongosari, and Q. Xie, "Automatic test generation from GUI applications for testing web services," in *ICSM: Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM 2007)*, pp. 345–354, Paris, France, Oct. 2007, IEEE Computer Society.

[30] B. A. Cota, D. G. Fritz, and R. G. Sargent, "Control flow graphs as a representation language," in *WSC '94: Proceedings of the 26th conference on Winter Simulation*, pp. 555–559, Orlando, Florida, United States, Dec. 1994, Society for Computer Simulation International.

[31] CPNTOOLS, [Online] http://wiki.daimi.au.dk/cpntools/cpntools.wiki.

[32] U. Dahan, "Autonomous services and enterprise entity aggregation." [Online], http://msdn.microsoft.com/en-us/library/bb245672.aspx ,July 2006.

[33] G. Dai, X. Bai, Y. Wang, and F. Dai, "Contract-based testing for web services," in *COMPSAC 2007: Proceedings of the 31st Annual International Computer Software and Applications Conference*, vol. 1, pp. 517–526, Beijing, China, July 2007, IEEE Computer Society.

[34] G. Dai, X. Bai, and C. Zhao, "A framework for model checking web service compositions based on bpel4ws," in *ICEBE 2007: Proceedings of the IEEE International Conference on e-Business Engineering*, pp. 165–172, Hong Kong, China, Oct. 2007, IEEE Computer Society.

[35] S. Dalal, A. Jain, N. Karunanithi, J. Leaton, C. Lott, G. Patton, and B. Horowitz, "Model-based testing in practice," in *ICSE '99: Proceedings of the International Conference on Software Engineering*, pp. 285–294, Los Angeles, California, USA, May 1999.

[36] L. F. J. de Almeida and S. R. Vergilio, "Exploring perturbation based testing for web services," in *ICWS '06: Proceedings of the 2006 IEEE International Conference on Web Services*, pp. 717–726, Chicago, IL, USA, 2006, IEEE Computer Society.

[37] M. Di Penta, M. Bruno, G. Esposito, V. Mazza, and G. Canfora, "Web services regression testing," in *Test and Analysis of Web Services* (L. Baresi and E. D. Nitto, eds.), pp. 205–234, Springer, 2007.

[38] Disjunctive Normal Form, [Online] http://mathworld.wolfram.com/DisjunctiveNormalForm.html.

[39] W.-L. Dong and H. YU, "Web service testing method based on fault-coverage," in *EDOC Workshops: The 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*, pp. 43–43, Hong Kong, China, Oct. 2006, IEEE Computer Society.

[40] W.-L. Dong, H. Yu, and Y.-B. Zhang, "Testing BPEL-based web service composition using high-level petri nets," in *EDOC'06: The 10th IEEE International Enterprise Distributed Object Computing Conference*, pp. 441–444, Hong Kong, China, Oct. 2006, IEEE Computer Society.

[41] P. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Addison-Wesley Professional, 2007.

[42] I. K. El-Far, "Enjoying the perks of model-based testing," in *STARWEST 2001: Proceedings of the Software Testing, Analysis, and Review Conference*, San Jose, CA, USA, Oct. 2001.

[43] A. T. Endo, A. S. Simão, S. R. S. Souza, and P. S. L. Souza, "Web services composition testing: A strategy based on structural testing of parallel programs," in *TAIC-PART '08: Proceedings of the Testing: Academic & Industrial Conference - Practice and Research Techniques*, pp. 3–12, Windsor, UK, Aug. 2008, IEEE Computer Society.

[44] T. Erl, *SOA Principles of Service Design*. Prentice Hall PTR, 2007.

[45] ETSI ES 201 873-1, "The Testing and Test Control Notation version 3, Part1: TTCN-3 Core notation, V2.1.1," June 2005, [Online] http://www.ttcn-3.org/StandardSuite.htm.

[46] M. Felderer, R. Breu, J. Chimiak-Opoka, M. Breu, and F. Schupp, "Concepts for model-based requirements testing of service oriented systems," in *Software Engineering, SE 2009*, pp. 152–157, IASTED, Innsbruck, Austria, Mar. 2009.

[47] Finite State Machine, [Online] http://www.nist.gov/dads/HTML/finiteStateMachine.html.

[48] FIT: Framework for Integrated Test, [Online] http://fit.c2.com/.

[49] C. Fu, B. G. Ryder, A. Milanova, and D. Wonnacott, "Testing of java web services for robustness," in *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pp. 23–34, Boston, Massachusetts, USA, July 2004, ACM.

[50] X. Fu, T. Bultan, and J. Su, "Analysis of interacting BPEL web services," in *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pp. 621–630, New York, New York, USA, May 2004, ACM.

[51] J. García-Fanjul, C. de la Riva, and J. Tuya, "Generating test cases specifications for compositions of web services," in *Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)* (A. Bertolino and A. Polini, eds.), pp. 83–94, Palermo, Italy, June 9th 2006.

[52] J. García-Fanjul, C. de la Riva, and J. Tuya, "Generation of conformance test suites for compositions of web services using model checking," in *TAIC PART'06: Proceedings of Testing: Academic & Industrial Conference Practice and Research Techniques*, pp. 127–130, Windsor, UK, Aug. 2006, IEEE Computer Society.

[53] E. Grishikashvili, D. Reilly, N. Badr, and A. Taleb-Bendiab, "From component-based to service-based distributed applications assembly and management," in *EUROMICRO: The 29th Euromicro Conference*, pp. 99–106, Antalya, Turkey, Sept. 2003, IEEE Computer Society.

[54] Z. Guangquan, R. Mei, and Z. Jun, "A business process of web services testing method based on uml2.0 activity diagram," in *IITA'07: Proceedings of the Workshop on Intelligent Information Technology Application*, pp. 59–65, Nanchang, China, Dec. 2007, IEEE Computer Society.

[55] S. Hanna and M. Munro, "Fault-based web services testing," in *ITGN: 5th International Conference on Information Technology: New Generations (ITNG 2008)*, pp. 471–476, Las Vegas, NV, USA, April 2008, IEEE Computer Society.

[56] M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. A. Spoon, and A. Gujarathi, "Regression test selection for java software," in *OOPSLA '01: Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*, pp. 312–326, Tampa Bay, Florida, USA, 2001, ACM.

[57] R. Heckel and M. Lohmann, "Towards contract-based testing of web services," in *Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004)*, vol. 116, pp. 145–156, Barcelona, Spain, March 2005. Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004).

[58] High-level Petri Nets - Concepts, Definitions and Graphical Notation, [Online] http://www.petrinets.info/docs/pnstd-4.7.1.pdf.

[59] S. S. Hou, L. Zhang, Q. Lan, H. Mei, and J. S. Sun, "Generating effective test sequences for BPEL testing," in *QSIC 2009: Proceedings of the 9th International Conference on Quality Software*, Jeju, Korea, August 2009, IEEE Computer Society Press.

[60] S.-S. Hou, L. Zhang, T. Xie, and J.-S. Sun, "Quota-constrained test-case prioritization for regression testing of service-centric systems," in *ICSM: 24th IEEE International Conference on Software Maintenance (ICSM 2008)*, pp. 257–266, Beijing, China, Oct. 2008, IEEE.

[61] HP Service Test, [Online] http://h71028.www7.hp.com/enterprise/cache/19054-0-0-225-121.html.

[62] H. Huang, W.-T. Tsai, R. A. Paul, and Y. Chen, "Automated model checking and testing for composite web services," in *ISORC: 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pp. 300–307, Seattle, WA, USA, May 2005, IEEE Computer Society.

[63] H. Y. Huang, H. H. Liu, Z. J. Li, and J. Zhu, "Surrogate: A simulation apparatus for continuous integration testing in service oriented architecture," in *IEEE SCC: 2008 IEEE International Conference on Services Computing (SCC 2008)*, vol. 2, pp. 223–230, Honolulu, Hawaii, USA, July 2008, IEEE Computer Society.

[64] International Data Corporation (IDC), [Online] http://www.idc.com/.

[65] Java PathFinder, [Online] http://javapathfinder.sourceforge.net/.

[66] JESS rule engine, [Online] http://www.jessrules.com/jess/index.shtml.

[67] Y. Jiang, S.-S. Hou, J.-H. Shan, L. Zhang, and B. Xie, "Contract-based mutation for testing components," in *ICSM: 21st IEEE International Conference on Software Maintenance (ICSM'05)*, pp. 483–492, Budapest, Hungary, Sept. 2005, IEEE Computer Society.

[68] S. Jones, "Toward an acceptable definition of service [service-oriented architecture]," *IEEE Software*, vol. 22, pp. 87–93, May-June 2005.

[69] K. Kaschner and N. Lohmann, "Automatic test case generation for interacting services," in *Service-Oriented Computing - ICSOC 2008 Workshops: ICSOC 2008 International Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers* (G. Feuerlicht and W. Lamersdorf, eds.), vol. 5472 of *Lecture Notes in Computer Science*, pp. 66–78, Sydney, Australia, April 2009, Springer-Verlag.

[70] J. C. King, "Symbolic execution and program testing," *Communications of the ACM*, vol. 19, no. 7, pp. 385–394, 1976.

[71] M. Lallali, F. Zaidi, and A. Cavalli, "Timed modeling of web services composition for automatic testing," in *SITIS '07: Proceedings of the 2007 International IEEE Conference on Signal-Image Technologies and Internet-Based System*, pp. 417–426, Shanghai, China, Dec. 2007, IEEE Computer Society.

[72] M. Lallali, F. Zaidi, A. Cavalli, and I. Hwang, "Automatic timed test case generation for web services composition," in *ECOWS '08: Proceedings of the 2008 6th European Conference on Web Services*, pp. 53–62, Dublin, Ireland, Nov. 2008, IEEE Computer Society.

[73] N. Laranjeiro, S. Canelas, and M. Vieira, "wsrbench: An on-line tool for robustness benchmarking," in *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*, pp. 187–194, Honolulu, HI, USA, 2008, IEEE Computer Society.

[74] G. Laycock, *The Theory and Practice of Specification Based Software Testing*. PhD thesis, University of Sheffield, 2003.

[75] S. Lee, X. Bai, and Y. Chen, "Automatic mutation testing and simulation on OWL-S specified web services," in *ANSS-41 '08: Proceedings of the 41st Annual Simulation Symposium*, pp. 149–156, Ottawa, Canada, April 2008, IEEE Computer Society.

[76] B. Legeard, "BZ-Testing-Tools: Model-based test generator," in *The 18th IEEE International Conference on Automated Software Engineering (ASE 2003) - Demo Paper*, Montreal, Canada, Oct. 2003.

[77] C. Lenz, J. Chimiak-Opoka, and R. Breu, "Model Driven Testing of SOA–based software," in *Proceedings of the Workshop on Software Engineering Methods for Service-oriented Architecture (SEMSOA 2007)* (D. Lübke, ed.), pp. 99–110, Leibniz Universität Hannover, FG Software Engineering, Hannover, Germany, May 2007.

[78] L. Li, W. Chou, and W. Guo, "Control flow analysis and coverage driven testing for web services," in *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*, pp. 473–480, Beijing, China, Sept. 2008, IEEE Computer Society.

[79] Z. Li, W. Sun, Z. B. Jiang, and X. Zhang, "BPEL4WS unit testing: framework and implementation," in *ICWS '05: Proceedings of the 2005 IEEE International Conference on Web Services*, pp. 103–110 vol.1, Orlando, FL, USA, July 2005, IEEE Computer Society.

[80] Z. J. Li and T. Maibaum, "An approach to integration testing of object-oriented programs," in *QSIC '07: Proceedings of the Seventh International Conference on Quality Software*, pp. 268–273, Portland, OR, USA, Oct. 2007, IEEE Computer Society.

[81] Z. J. Li, J. Zhu, L.-J. Zhang, and N. Mitsumori, "Towards a practical and effective method for web services test case generation," in *Proceedings of the ICSE Workshop on Automation of Software Test (AST'09)*, pp. 106–114, May 2009.

[82] D. Liang and K. Xu, "Testing scenario implementation with behavior contracts," in *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*, vol. 1, pp. 395–402, Chicago, IL, USA, Sept. 2006, IEEE Computer Society.

[83] F. Lin, M. Ruth, and S. Tu, "Applying safe regression test selection techniques to java web services," in *NWESP '06: Proceedings of the International Conference on Next Generation Web Services Practices*, pp. 133–142, Seoul, South Korea, Sept. 2006, IEEE Computer Society.

[84] F. Liu, F. yuan Ma, , M. lu Li, and L. peng Huang, *Web Information Systems – WISE 2004 Workshops*, ch. A Framework for Semantic Grid Service Discovery, pp. 3–10. SpringerLink, 2004.

[85] H. Liu, Z. Li, J. Zhu, and H. Tan, *Service-Oriented Computing ICSOC 2007*, vol. 4749/2009, ch. Business Process Regression Testing, pp. 157–168. Springer-Verlag Berlin Heidelberg, 2007.

[86] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing interacting BPEL processes," in *Forth International Conference on Business Process Management (BPM 2006)* (S. Dustdar, J. Fiadeiro, and A. Sheth, eds.), vol. 4102 of *Lecture Notes in Computer Science*, p. 1732, Vienna, Austria, September 2006, Springer-Verlag.

[87] N. Looker, J. Xu, and M. Munro, "Determining the dependability of service-oriented architectures," *International Journal of Simulation and Process Modelling*, vol. 3, pp. 88–97, Jul. 2007.

[88] B. Lublinsky, "Use service-oriented decomposition to meet your architectural goals." [Online], http://www.ibm.com/developerworks/library/ar-soadecomp/ ,October 2007.

[89] C. Ma, C. Du, T. Zhang, F. Hu, and X. Cai, "WSDL-based automated test data generation for web service," in *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*, pp. 731–737, Wuhan, China, Dec. 2008, IEEE Computer Society.

[90] C. Ma, J. Wu, T. Zhang, Y. Zhang, and X. Cai, "Testing BPEL with Stream X-Machine," in *ISISE '08: Proceedings of the 2008 International Symposium on Information Science and Engieering*, pp. 578–582, Shanghai, China, Dec. 2008, IEEE Computer Society.

[91] E. Martin, S. Basu, and T. Xie, "Automated testing and response analysis ofweb services," in *ICWS '07: Proceedings of the 2007 IEEE International Conference on Web Services*, pp. 647–654, Salt Lake City, UT, USA, July 2007, IEEE Computer Society.

[92] P. Mayer and D. Lübke, "Towards a BPEL unit testing framework," in *TAV-WEB '06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pp. 33–42, Portland, Maine, USA, 2006, ACM.

[93] P. McMinn, "Search-based software test data generation: A survey," *Software Testing, Verification & Reliability (STVR)*, vol. 14, no. 2, pp. 105–156, 2004.

[94] H. Mei and L. Zhang, "A framework for testing web services and its supporting tool," in *SOSE '05: Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering*, pp. 199–206, Beijing, China, Oct. 2005, IEEE Computer Society.

[95] L. Mei, W. Chan, and T. Tse, "Data flow testing of service-oriented workflow applications," in *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pp. 371–380, Leipzig, Germany, May 2008, ACM.

[96] L. Mei, W. K. Chan, T. H. Tse, and R. G. Merkel, "Tag-based techniques for black-box test case prioritization for service testing," in *Proceedings of the 9th International Conference on Quality Software (QSIC 2009)*, Jeju, Korea, August 2009, IEEE Computer Society Press.

[97] METEOR-S, [Online] http://lsdis.cs.uga.edu/projects/meteor-s/.

[98] B. Meyer, "Applying 'Design by Contract'," *Computer*, vol. 25, pp. 40–51, Oct 1992.

[99] L. J. Morell, "A theory of fault-based testing," *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 844–857, 1990.

[100] S. Morimoto, "A survey of formal verification for business process modeling," in *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part II*, pp. 514–522, Kraków, Poland, June 2008, Springer-Verlag.

[101] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr 1989.

[102] M. Narita, M. Shimamura, K. Iwasa, and T. Yamaguchi, "Interoperability verification for web service based robot communication platforms," in *ROBIO 2007: Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetics*, pp. 1029–1034, Sanya, China, Dec. 2007, IEEE Computer Society.

[103] C. Nebut, F. Fleurey, Y. L. Traon, and J.-M. Jézéquel, "Requirements by contracts allow automated system testing," in *ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering*, pp. 85–96, Denver, CO, USA, Nov. 2003, IEEE Computer Society.

[104] NuSMV, [Online] http://nusmv.irst.itc.it/.

[105] J. Offutt and W. Xu, "Generating test cases for web services using data perturbation," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, pp. 1–10, 2004.

[106] Oracle Application Testing Suite, [Online] http://www.oracle.com/enterprise/_manager/application-quality-solutions.html.

[107] C. Ouyang, H. M. W. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. H. M. ter Hofstede, *Service-Oriented Computing - ICSOC 2005*, ch. WofBPEL: A Tool for Automated Analysis of BPEL Processes, pp. 484–489. Springerlink, 2005.

[108] OWL-S: Semantic Markup for Web Services, [Online] http://www.w3.org/Submission/OWL-S/.

[109] Oxford English Dictionary 2nd. ed., Oxford University Press, 1989 [Online] http://dictionary.oed.com/cgi/entry/50015226.

[110] M. P. Papazoglou, "JDL special issue on service-oriented computing: Advanced user-centered concepts," *International Journal on Digital Libraries*, vol. 6, pp. 233–234, 2006.

[111] M. P. Papazoglou, "Introduction to special issue on service oriented computing (SOC)," *ACM Transactions on the Web*, vol. 2, no. 2, pp. 1–2, 2008.

[112] M. P. Papazoglou and J. jacques Dubray, "A survey of web service technologies," tech. rep., University of Trento, Jun. 2004.

[113] A. Paradkar, A. Sinha, C. Williams, R. Johnson, S. Outterson, C. Shriver, and C. Liang, "Automated functional conformance test generation for semantic web services," in *ICWS '07: Proceedings of the 2007 IEEE International Conference on Web Services*, pp. 110–117, Salt Lake City, UT, USA, July 2007.

[114] C. Pautasso, "JOpera: An agile environment for web service composition with visual unit testing and refactoring," in *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 311–313, Dallas, TX, USA, Sept. 2005, IEEE Computer Society.

[115] C. Pautasso and G. Alonso, "The JOpera visual composition language," *Journal of Visual Languages and Computing (JVLC)*, vol. 16, no. 1-2, pp. 119–152, 2005.

[116] Petri Net Markup Language (PNML), [Online] http://www2.informatik.hu-berlin.de/top/pnml/about.html.

[117] L. Peyton, B. Stepien, and P. Seguin, "Integration testing of composite applications," in *HICSS '08: Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, pp. 96–96, Waikoloa, Big Island, Hawaii, Jan. 2008, IEEE Computer Society.

[118] D. Pilone and N. Pitman, *UML 2.0 in a Nutshell (In a Nutshell (O'Reilly))*. O'Reilly Media, Inc., 2005.

[119] Promela Manual, [Online] http://spinroot.com/spin/Man/promela.html.

[120] Publish/Subscribe Model, [Online] http://msdn.microsoft.com/en-us/library/ms978603.aspx.

[121] P. Ramsokul and A. Sowmya, "ASEHA: A framework for modelling and verification of web services protocols," in *SEFM '06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pp. 196–205, Pune, India, Sept. 2006, IEEE Computer Society.

[122] P. Ramsokul and A. Sowmya, "A sniffer based approach to ws protocols conformance checking," in *ISPDC '06: Proceedings of the Proceedings of The Fifth International Symposium on Parallel and Distributed Computing*, pp. 58–65, Timisoara, Romania, July 2006, IEEE Computer Society.

[123] O. Rana, A. Akram, R. A. Ali, D. Walker, G. von Laszewski, and K. Amin, *Extending Web Services Technologies*, ch. Quality-of-Service Based Grid Communities, pp. 161–186. SpringerLink, 2004.

[124] Resource Description Framework (RDF), [Online] http://www.w3.org/RDF/.

[125] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, no. 2, pp. 173–210, 1997.

[126] M. Ruth, S. Oh, A. Loup, B. Horton, O. Gallet, M. Mata, and S. Tu, "Towards automatic regression test selection for web services," in *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference*, vol. 2, pp. 729–736, Beijing, China, July 2007.

[127] M. Ruth and S. Tu, "Concurrency issues in automating RTS for web services," in *ICWS '07: Proceedings of the 2007 IEEE International Conference on Web Services*, pp. 1142–1143, Salt Lake City, UT, USA, July 2007, IEEE Computer Society.

[128] M. Ruth and S. Tu, "A safe regression test selection technique for web services," in *ICIW '07: Proceedings of the Second International Conference on Internet and Web Applications and Services*, pp. 47–47, Mauritius, May 2007, IEEE Computer Society.

[129] H. Schlingloff, A. Martens, and K. Schmidt, "Modeling and model checking web services," *Electronic Notes in Theoretical Computer Science*, vol. 126, pp. 3–26, 2005. Proceedings of the 2nd International Workshop on Logic and Communication in Multi-Agent Systems (2004).

[130] K. Schmidt, "LoLA: A Low Level Analyser," in *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000)*, vol. 1825/2000 of *Lecture Notes in Computer Science*, pp. 465–474, Aarhus, Denmark, June 2000, Springer Berlin / Heidelberg.

[131] Semantic Web Services Language (SWSL), [Online] http://www.daml.org/services/swsl/.

[132] K. Senthil Kumar, A. S. Das, and S. Padmanabhuni, "WS-I Basic Profile: A practitioner's view," in *ICWS '04: Proceedings of the 2004 IEEE International Conference on Web Services*, pp. 17–24, San Diego, CA, USA, July 2004, IEEE Computer Society.

[133] R. Shukla, D. Carrington, and P. Strooper, "A passive test oracle using a component's API," in *APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference*, pp. 561–567, Tapei, Taiwan, Dec. 2005, IEEE Computer Society.

[134] R. Siblini and N. Mansour, "Testing web services," in *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications*, p. 135, Cairo, Egypt, Jan. 2005, IEEE Computer Society.

[135] A. Sinha and A. Paradkar, "Model-based functional conformance testing of web services operating on persistent data," in *TAV-WEB '06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications*, pp. 17–22, Portland, Maine, 2006, ACM.

[136] C. Smythe, "Initial investigations into interoperability testing of web services from their specification using the unified modelling language," in *Proceedings of the International Workshop on Web Services Modeling and Testing (WS-MaTe2006)* (A. Bertolino and A. Polini, eds.), pp. 95–119, Palermo Italy, June 9th 2006.

[137] H. M. Sneed and S. Huang, "WSDLTest - a tool for testing web services," in *WSE '06: Proceedings of the Eighth IEEE International Symposium on Web Site Evolution*, pp. 14–21, Philadelphia, PA, USA, Sept. 2006, IEEE Computer Society.

[138] SOAP Sonar, [Online] http://www.crosschecknet.com/.

[139] SOAP Version 1.2, [Online] http://www.w3.org/TR/soap12-part1/.

[140] SOATest, [Online] http://www.parasoft.com/.

[141] SPIN, [Online] http://spinroot.com/spin/whatispin.html.

[142] SWRL: A Semantic Web Rule Language, [Online] http://www.w3.org/Submission/SWRL/.

[143] A. Tarhini, H. Fouchal, and N. Mansour, "A simple approach for testing web service based applications," in *Proceedings of the 5th International Workshop on Innovative Internet Community Systems (IICS 2005)* (A. Bui, M. Bui, T. Böhme, and H. Unger, eds.), vol. 3908 of *Lecture Notes in Computer Science*, pp. 134–146, Paris, France, 2005, Springer.

[144] A. Tarhini, H. Fouchal, and N. Mansour, "Regression testing web services-based applications," in *Proceedings of the 4th ACS/IEEE International Conference on Computer Systems and Applications*, pp. 163–170, Sharjah, UAE, Mar. 2006, IEEE Computer Society.

[145] W. T. Tsai, X. Bai, Y. Chen, and X. Zhou, "Web service group testing with windowing mechanisms," in *SOSE '05: Proceedings of the IEEE International Workshop*, pp. 221–226, Beijing, China, Oct. 2005, IEEE Computer Society.

[146] W. T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang, and R. A. Paul, "Testing web services using progressive group testing," in *Advanced Workshop on Content Computing (AWCC 2004)* (C.-H. Chi and K.-Y. Lam, eds.), vol. 3309 of *Lecture Notes in Computer Science*, pp. 314–322, Zhenjiang, Jiangsu, China, 2004, Springer.

[147] W. T. Tsai, Y. Chen, R. Paul, H. Huang, X. Zhou, and X. Wei, "Adaptive testing, oracle generation, and test case ranking for web services," in *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference*, vol. 2, pp. 101–106, Edinburgh, UK, July 2005, IEEE Computer Society.

[148] W. T. Tsai, Y. Chen, R. Paul, N. Liao, and H. Huang, "Cooperative and group testing in verification of dynamic composite web services," in *COMPSAC '04: Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts*, vol. 2, pp. 170–173, Hong Kong, China, Sept. 2004, IEEE Computer Society.

[149] W. T. Tsai, Y. Chen, D. Zhang, and H. Huang, "Voting multi-dimensional data with deviations for web services under group testing," in *ICDCSW '05: Proceedings of the 4th International Workshop on Assurance in Distributed Systems and Networks (ADSN)*, pp. 65–71, Columbus, OH, USA, June 2005, IEEE Computer Society.

[150] W. T. Tsai, M. Malek, Y. Chen, and F. Bastani, "Perspectives on service-oriented computing and service-oriented system engineering," in *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering*, pp. 3–10, Shanghai, China, Oct. 2006, IEEE Computer Society.

[151] W. T. Tsai, R. Paul, Z. Cao, L. Yu, and A. Saimi, "Verification of web services using an enhanced UDDI server," in *Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, pp. 131–138, Turku, Finland, Jan. 2003.

[152] W. T. Tsai, R. Paul, W. Song, and Z. Cao, "Coyote: An XML-based framework for web services testing," in *HASE'02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering*, p. 173, Tokyo, Japan, Oct. 2002, IEEE Computer Society.

[153] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang, "Extending WSDL to facilitate web services testing," in *HASE'02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering*, pp. 171–172, Tokyo, Japan, Oct. 2002.

[154] W. T. Tsai, X. Wei, Y. Chen, and R. Paul, "A robust testing framework for verifying web services by completeness and consistency analysis," in *SOSE '05: Proceedings of the IEEE International Workshop*, pp. 151–158, Beijing, China, Oct. 2005, IEEE Computer Society.

[155] W. T. Tsai, X. WEI, Y. Chen, R. Paul, and B. Xiao, "Swiss cheese test case generation for web services testing," *IEICE - Transactions on Information and Systems*, vol. E88-D, no. 12, pp. 2691–2698, 2005.

[156] W. T. Tsai, X. Zhou, R. Paul, Y. Chen, and X. Bai, "A coverage relationship model for test case selection and ranking for multi-version software," in *HASE '07: Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium*, pp. 105–112, Dallas, TX, USA, Nov. 2007, IEEE Computer Society.

[157] UDDI Spec Technical Committee Draft, [Online] http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.

[158] M. Vieira, N. Laranjeiro, and H. Madeira, "Benchmarking the robustness of web services," in *PRDC '07: Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*, pp. 322–329, Melbourne, Victoria, Australia, Dec. 2007, IEEE Computer Society.

[159] D. Wang, B. Li, and J. Cai, "Regression testing of composite service: An XBFG-based approach," in *Proceedings of the 2008 IEEE Congress on Services Part II (SERVICES-2 '08)*, pp. 112–119, Beijing, China, 2008, IEEE Computer Society.

[160] R. Wang and N. Huang, "Requirement model-based mutation testing for web service," in *NWESP '08: Proceedings of the 2008 4th International Conference on Next Generation Web Services Practices*, pp. 71–76, Seoul, Korea, Oct. 2008, IEEE Computer Society.

[161] Y. Wang, X. Bai, J. Li, and R. Huang, "Ontology-based test case generation for testing web services," in *ISADS '07: Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems*, pp. 43–50, Sedona, AZ, USA, Mar. 2007, IEEE Computer Society.

[162] Web Service Interoperability Organisation (WS-I), "Basic Profile 1.2," [Online] http://www.ws-i.org /deliverables/workinggroup.aspx?wg=basicprofile.

[163] Web Service Interoperability Organization (WS-I), [Online] http://www.ws-i.org/.

[164] Web Service Modelling Ontology (WSMO), [Online] http://www.wsmo.org/.

[165] Web Service Semantics (WSDL-S), [Online] http://www.w3.org/Submission/WSDL-S/.

[166] Web Services Addressing (WS-Addressing), [Online] http://www.w3.org/Submission/ws-addressing/.

[167] Web Services Atomic Transaction (WS-AtomicTransaction), [Online] http://schemas.xmlsoap.org/ws/2004/10/wsat/.

[168] Web Services Business Activity (WS-BusinessActivity), [Online] http://docs.oasis-open.org/ws-tx/wsba/2006/ 06.

[169] Web Services Description Language (WSDL 1.1), [Online] http://www.w3.org/TR/wsdl.

[170] Web Services Glossary, [Online] http://www.w3.org/TR/ws-gloss/.

[171] Web Services Metadata Exchange (WS-MetadataExchange), [Online] http://www.w3.org/TR/2009/WD-ws-metadata-exchange-20090317/.

[172] E. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Transactions on Software Engineering*, vol. 17, pp. 703–711, Jul 1991.

[173] WofBPEL and BPEL2PNML, [Online] http://www.bpm.fit.qut.edu.au/projects/babel/tools/.

[174] WSDL2Java, [Online] http://cwiki.apache.org/CXF20DOC/wsdl-to-java.html.

[175] wsrbench, [Online] http://wsrbench.dei.uc.pt/.

[176] J. O. Wuzhi Xu and J. Luo, "Testing web services by XML perturbation," in *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, pp. 257–266, Chicago, IL, USA, 2001, IEEE Computer Society.

[177] XML Path Language (XPath), [Online] http://www.w3.org/TR/xpath/.

[178] C. Xu, H. Wang, and W. Qu, "Modeling and verifying BPEL using synchronized net," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pp. 2358–2362, Fortaleza, Ceara, Brazil, March 2008, ACM.

[179] W. Xu, J. Offutt, and J. Luo, "Testing web services by XML perturbation," in *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, pp. 257–266, Chicago, IL, USA, Nov. 2005, IEEE Computer Society.

[180] J. Yan, Z. Li, Y. Yuan, W. Sun, and J. Zhang, "BPEL4WS unit testing: Test case generation using a concurrent path analysis approach," in *ISSRE '06: Proceedings of the 17th International Symposium on Software Reliability Engineering*, pp. 75–84, Raleigh, NC, USA, Nov. 2006, IEEE Computer Society.

[181] Y. Yang, Q. Tan, and Y. Xiao, "Verifying web services composition based on hierarchical colored petri nets," in *IHIS '05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pp. 47–54, Bremen, Germany, Nov. 2005, ACM.

[182] X. Yi and K. Kochut, "A CP-nets-based design and verification framework for web services composition," in *ICWS '04: Proceedings of the 2004 IEEE International Conference on Web Services*, pp. 756–760, San Diego, CA, USA, July 2004, IEEE Computer Society.

[183] Y. Yu, N. Huang, and Q. Luo, "OWL-S based interaction testing of web service-based system," in *NWESP '07: Proceedings of the Third International Conference on Next Generation Web Services Practices*, pp. 31–34, Seoul, South Korea, Oct. 2007, IEEE Computer Society.

[184] Y. Yu, N. Huang, and M. Ye, "Web services interoperability testing based on ontology," in *CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology*, pp. 1075–1079, Binghamton, NY, USA, Sept. 2005, IEEE Computer Society.

[185] Y. Yuan, Z. Li, and W. Sun, "A graph-search based approach to BPEL4WS test generation," in *ICSEA '06: Proceedings of the International Conference on Software Engineering Advances*, p. 14, Tahiti, French Polynesia, Oct. 2006, IEEE Computer Society.

[186] J. Zhang and L.-J. Zhang, "Criteria analysis and validation of the reliability of web services-oriented systems," in *ICWS '05: Proceedings of the 2005 IEEE International Conference on Web Services*, pp. 621–628, Orlando, FL, USA, July 2005, IEEE Computer Society.

[187] Y. Zhang and H. Zhu, "Ontology for service oriented testing of web services," in *SOSE '08: Proceedings of the 2008 IEEE International Symposium on Service-Oriented System Engineering*, pp. 129–134, 2008, IEEE Computer Society.

[188] Y. Zheng, J. Zhou, and P. Krause, "Analysis of BPEL data dependencies," in *EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 351–358, Aug. 2007, IEEE Computer Society.

[189] Y. Zheng, J. Zhou, and P. Krause, "A model checking based test case generation framework for web services," in *ITNG '07: Proceedings of the International Conference on Information Technology*, pp. 715–722, Las Vegas, NV, USA, April 2007, IEEE Computer Society.

[190] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen, "Metamorphic testing and its applications," in *Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004)*, Xian, China, October 2004.

[191] H. Zhu, "A framework for service-oriented testing of web services," in *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*, vol. 2, pp. 145–150, Sept. 2006, IEEE Computer Society.