SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. (2012)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.1470

Testing and verification in service-oriented architecture: a survey

Mustafa Bozkurt, Mark Harman and Youssef Hassoun*,†

Centre for Research on Evolution, Search and Testing (CREST), Department of Computer Science, University College London, Malet Place, London WC1E 6BT, UK

SUMMARY

Service-oriented architecture (SOA) is gaining momentum as an emerging distributed system architecture for business-to-business collaborations. This momentum can be observed in both industry and academic research. SOA presents new challenges and opportunities for testing and verification, leading to an upsurge in research. This paper surveys the previous work undertaken on testing and verification of service-centric systems, which in total are 177 papers, showing the strengths and weaknesses of current strategies and testing tools and identifying issues for future work. Copyright © 2012 John Wiley & Sons, Ltd.

Received 20 January 2010; Revised 11 January 2012; Accepted 23 February 2012

1. INTRODUCTION

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 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.

Several software companies and market analysis institutions have highlighted the changes SOC brought. The focus of businesses is migrating from product manufacturing (hardware and software) to service provision. For example, according to AppLabs [1], 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 Wintergreen Research Inc. [2], the changes in the software industry go beyond technical issues. With software becoming more agile in order to fit business requirements, many businesses position towards adopting service-centric system(s) (ScS).

The market growth of SOC-related technology also supports this claim. Wintergreen Research Inc.'s estimation for the 2005 SOC market is $450m, and it is expected to reach $18.4bn in 2012. According to the International Data Corporation [3], a leading market research institution, the global spend on service-oriented software in 2006 was nearly $2bn. The same report projects that this spend will rise to $14bn in 2011.

However, when compared with optimistic predictions, SOC may yet to achieve its full market potential. The authors believe that this is due to two main reasons. The first reason is the recent economic downturn, which reduces investment in new technologies and research. The second reason is the technological challenges that are brought by ScS.

*Correspondence to: Youssef Hassoun, Department of Computer Science, University College London (UCL), Malet Place, London WC1E 6BT, UK.
†E-mail: youssef.hassoun@kcl.ac.uk

> 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 [4].

As stated by CDBI Forum [4], one of the main technological barriers to enterprises' transition to ScS is denoted by the heightened importance of the issue of trust. Web services are introduced into systems that require high reliability and security. Using services in these system raises the importance of establishing trust between the service provider and the consumer. The trust issue is not just about correct functioning of a service. It has many other dimensions such as service security and quality of service (QoS).

Testing provides one potential solution to the issue of establishing trust. Testing is important to assure the correct functioning of ScS, which, by nature, has the ability to dynamically select and use services. In order to confirm the correct functioning of an ScS, interoperability among all its components and integration of these components must be adequately tested.

Service-centric systems may also require more frequent testing than traditional software. The possibility of changes to an ScS increases with the number of services involved. With every change to an invoked service, an ScS might need testing. Testing ScS for changes presents other challenges, such as identifying the right time to test and operations that are affected by the changes made. These problems occur especially when changes are made to services provided by third parties.

Service-centric systems also need to be tested and monitored for QoS to ensure that they perform at expected levels. In order to compose a service-level agreement (SLA), an agreement between the service provider and the consumer on service operation, QoS parameters have to be measured and subsequently monitored for their compliance. This process is well-known practice in other fields such as Web hosting and network services. Because of the number of stakeholders involved and dynamic late binding in service-oriented architecture (SOA), establishing an SLA might not be as straightforward as those used in other fields. These problems also effect testing and monitoring for QoS in SOA.

As a result, ScS require more effective and efficient testing/monitoring than traditional software systems. Unfortunately, most of the existing testing approaches for distributed and agent-based systems are insufficient for SOA. According to Canfora and Di Penta [5], the issues that limit the testability of ScS are the following:

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; and
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; and
   (c) effects of testing in some systems, such as stock exchange systems, where each usage of the service means a business transaction.

In order to overcome these limitations, existing testing approaches have to be adapted to become sufficient for SOA. To help the need for efficient testing and monitoring specific test beds are also required in SOA. These test beds must be able to perform as many of the required testing methods as possible, and they also must be easy to deploy. Ease of deployment is an important feature because it usually takes a long time to set up a testing environment for a distributed system. However, such delays may not be acceptable in ScS testing (ScST) because of increased testing frequency.

The present publication presents a comprehensive survey of existing work in ScST. ScS present important challenges to software testers. The stated challenges in SOA have led to much work on techniques that are adapted for ScST.

DOI: 10.1002/stvr

A recent overview survey by Canfora and Di Penta [6] summarizes ScST and categorizes the research undertaken into four categories: functional, regression, integration and nonfunctional testing. The current survey extends Canfora and Di Penta's survey by classifying research undertaken in ScST 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 present survey follows an organizational flow that seeks to shadow the life cycle of ScS from development to publishing and versioning. The remainder of this survey is organized as follows. Section 2 explains perspectives in ScST. Section 3 discusses issues in ScST. Section 4 discusses the current trends in ScST. Section 5 explains test case generation techniques applied to web services. Section 6 discusses the issues related to unit testing of ScS and proposed solutions to these issues. Section 7 discusses fault-based testing of ScS. Section 8 examines model-based testing (MBT) and formal verification of ScS. Section 9 discusses interoperability testing of ScS. Section 10 discusses integration testing of ScS. Section 11 introduces the collaborative testing concept and describes approaches that use this concept. Section 12 discusses testing for QoS violations. Section 13 discusses regression testing of ScS. Section 14 discusses the current problems that require solutions. Section 15 concludes the survey.

2. SERVICE-CENTRIC SYSTEM TESTING PERSPECTIVES

As expected from a distributed environment, SOA has multiple stakeholders involved. Canfora and Di Penta [7] have specified five stakeholder in SOA. These stakeholders are the developer, the provider, the integrator, the third-party certifier and the user.

Developer: The developer is the stakeholder who implements the service and has the sole responsibility for the service evolution.

Provider: The provider is the stakeholder who deals with QoS in the SOA. The provider ensures that the service operates within the conditions defined by the SLA.

Integrator: The integrator is the stakeholder who uses existing services in a composition or an application.

Third-party certifier: The third-party certifier (the certifier in rest of this survey) is the stakeholder who provides testing services to the other stakeholders.

End-user: The end-user is the stakeholder who uses the service through an application or platform.

3. SERVICE-CENTRIC SYSTEM TESTING CHALLENGES

Service-centric system testing includes testing of the basic service functionality, service interoperability, some of the SOA functionalities, QoS and load/stress testing [8]. Canfora and Di Penta [7] discussed the challenges involved in testing different aspects of Web services and categorized these challenges.

Service-centric system testing is generally accepted to be more challenging compared with testing of traditional systems. It has been claimed that services bring new challenges to software testing [5, 9, 10]. These challenges include the following:

1. specification-based testing using all or some of the Web service specifications such as Web Service Description Language (WSDL) [11], Ontology Web Language for Services (OWL-S) [12] and Web Service Modeling Ontology (WSMO) [13];
2. run-time testing of SOA activities (discovery, binding and publishing) due to the dynamic nature of ScS;
3. testing in collaboration due to the multiple stakeholders involved in SOA activities; and
4. testing for web service selection, which requires testing services with same specifications but different implementations.

Service-centric system testing is challenging not because of the testing issues it brings that did not exist for traditional software but because of the limitations that the testers face in ScST. ScST is

DOI: 10.1002/stvr

more challenging compared with traditional systems for two primary reasons: the complex nature of Web services and the limitations that occur because of the nature of SOA. It has been argued [9, 14] that the distributed nature of Web services based on multiple protocols such as Universal Description Discovery and Integration (UDDI) [15] and Simple Object Access Protocol (SOAP) [16], together with the limited system information provided with WSDL specifications, makes ScST challenging. A similar but more comprehensive discussion to the issues that limit the testability of ScS was presented in Section 1. Issues 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 services bring to software testing, some existing testing techniques can be adapted to ScST [17]. Adaptation is mainly needed to overcome the factors that limit the capability of testing techniques. The degree of adaptation required 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 in ScST, and these approaches must be adapted to SOA.

4. SERVICE-CENTRIC SYSTEM TESTING TRENDS

In order to understand the trends in ScST, all the existing work in this field will be analysed from several different perspectives. To give better idea about the different aspects of this field, comparisons with different research fields are also presented.

The first and one of the most important aspects that proves the significance of a research field is the number of research publications. Figure 1 shows the total number of publications from 2002 to 2010. Total number of publications in 2010 might not reflect the actual number because of its recency.

Analysis of the research publications on the subject revealed its rapid growth. The trend line plotted on Figure 1 with coefficient determination value (R^2) $\tilde{0}.995$ is very assuring to make future predictions. The trend line suggests that, if this trend continues until 2012, the number of publications will be around 250, close to twice the number of publications in 2009.

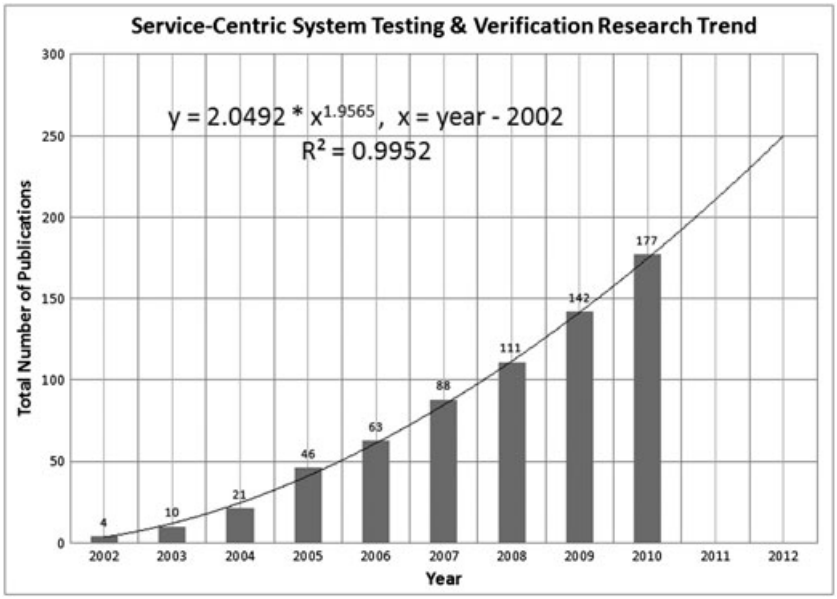

Figure 1. Total number of publications from 2002 to 2010.

DOI: 10.1002/stvr

Furthermore, the present survey only focusses on functional ScST. The total number of research publications in ScST can be much higher than that stated because the present survey does not include other important testing areas such as security testing and performance testing.

Analysis of the publication volume for the testing techniques and methodologies used in ScST is presented in Figures 2 and 3. Formal verification, MBT and fault-based testing are the three on which the larger volumes are noted.

Existing information on the application of ScST in industry allows for some comparison of the evolution of ScST in industry and academia. According to Bloomberg [18], the history of Web service testing is divided into three phases, on the basis of the functionalities that are added to ScST 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 and binding capabilities of SOA, the asynchronous 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 run-time capabilities of web services were tested. Testing web service compositions and service versioning testing emerged during this period.

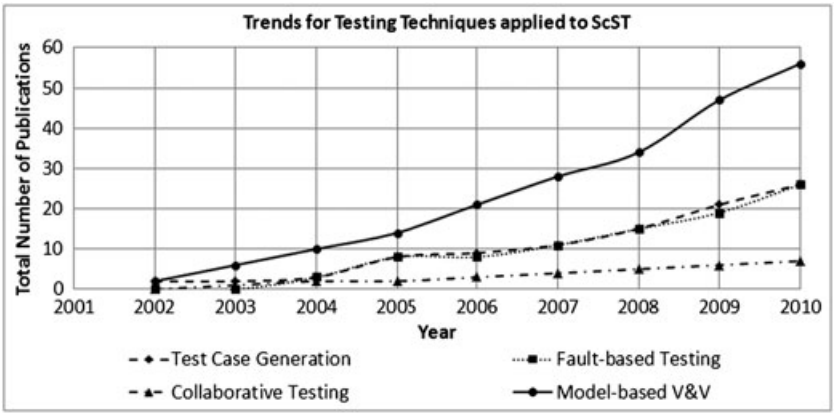

Figure 2. Publication trends of testing techniques applied to Service-centric system testing.

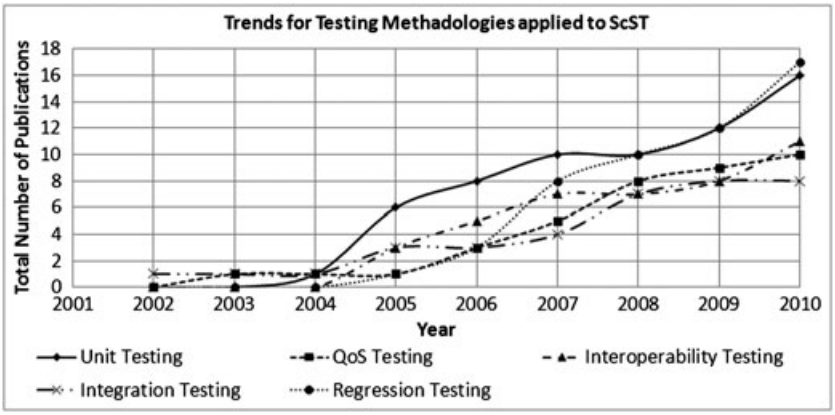

Figure 3. Publication trends of testing methodologies applied to Service-centric system testing.

DOI: 10.1002/stvr

Analysis of the academic work in Figures 2 and 3 do not yield results similar to Bloomberg's analysis. For example, in the 2002–2003 period, five publications on formal verification, one on test data generation, one on integration, one on QoS and one on collaborative testing were published. In academic research, the first work using unit testing appeared in 2005. Analysis for the remaining periods also shows a relatively small apparent correlation between the industry and academia.

The final analysis on existing research covers the experiments performed in the publications to validate the applicability and the effectiveness of the proposed approaches. This analysis gives an idea of the case studies used by others. Figure 4 presents the type of case studies used by the researchers.

This figure highlights one of the great problems in ScST research: lack of real-world case studies. Unfortunately, 71% of the research publications provide no experimental results. The synthetic service portion of the graph mainly includes approaches for testing Business Process Execution Language (BPEL) compositions. The most common examples are the loan approval and the travel agency (travel booking) systems that are provided with BPEL engines. There are also experiments on existing Java code that are turned into services such as javaDNS, JNFS and Haboob. The real services used in experiments are generally existing small public services that can be found on the Internet. There are four experiments performed on existing projects such as government projects, Mars communication service and a human resources (HR) system.

5. TEST CASE GENERATION FOR WEB SERVICES

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 system/service under test (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. This section focusses on the specification-based test data generation, and model-based approaches are discussed in Section 8.

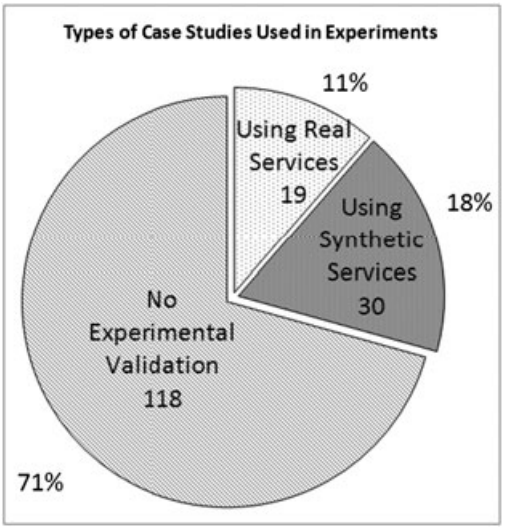

Figure 4. Distribution of case study types used in experimental validation. Number within the circle indicates absolute number of papers.

DOI: 10.1002/stvr

Test cases are not only used for detecting the faults in system behaviour but are also used for proving the nonexistence of known errors. Fault-based test data generation aims to prove the absence of prescribed faults in services. Unlike in regular test data generation, in fault-based test data generation, erroneous test data is generated intentionally. Test data generation for fault-based testing is discussed in detail in Section 7.

In order to retain clear separation between testing strategies that only involve Web services and the ScST strategies that aim to test various aspects of SOA (including services), we introduce the term Web service testing (WST). WST is a subset of ScST.

Work on generating test cases for WST can be grouped into three categories: specification-based test case generation, contract-based test case generation and partition testing. The approaches that use these techniques are presented separately in order to provide the reader a clearer comparison among the approaches using the same technique. However, they are discussed together in order to highlight their contributions to test case generation in WST.

5.1. Perspectives in test case generation

Test data generation is usually classified into two categories black-box (functional) and white-box (structural) testing. Black-box approaches are mainly used by the users of a software system who does not have the source code, whereas white-box approaches are used by the developer.

In SOA, as explained in Section 2, the developer is the only stakeholder who has access to the source code at the service level and can generate test cases by using white-box approaches. The rest of the stakeholders have only access to the service specifications and can only use black-box approaches. If the testing is performed at the composition level, the integrator is the only stakeholder who can generate structural tests. Test case generation approaches for the integrator are discussed in Section 8.

The aim of contracts in software testing is to increase testability. Contracts can help increase the effectiveness of the testing process for the tester by providing more information on the SUT's expected behaviour. Contract-based testing can be especially useful to the integrator and the certifier. In contract-based testing, it is expected that the developer provides the required contracts. As a result, contracts increase the cost of service development because of the time and effort it takes to create them.

Partition testing can be performed by any of the SOA stakeholders. Because the aim of the partition testing is to reduce the cost of testing by reducing the number of test cases, it might be especially useful for the integrator and the certifier.

5.2. Test case generation approaches

This section is divided into three groups on the basis of the test data generation method used.

5.2.1. Specification-based test case 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 or a user manual. Naturally, in specification-based testing, test cases are generated using the available system specifications.

In SOA, the first information the tester receives about a service is its specification. In this situation, specification-based testing becomes a natural choice. Test case generation for web services, as expected, is based on the service specifications. For traditional Web services, the provided WSDL specifications include abstract information on the available operations and their parameters. Information from WSDL specification allows generation of test cases for boundary value analysis, equivalence class testing or random testing using the XML schema data type information.

Many proposed approaches [9, 19–25] for WSDL-based test case generation are based on the XML schema information. Test cases for each service operation are generated on the basis of different coverage criteria such as operation coverage, message coverage and operation flow coverage.

Input data for the test cases can also be generated using schema information. The data type information with various constraints for each data type allows generation of test data for each simple

DOI: 10.1002/stvr

type. In XML, complex data types can also be defined. Test data generation for complex data types simply requires decomposition of the complex type into simple types. Test data is generated for each of these simple types and the combined data is used as complex test data.

Li *et al.* [22] proposed a test case generation method that combines information from WSDL specifications and user knowledge. Li *et al.* introduced a tool called WSTD-Gen that supports this method. The tool allows users to customize data types and select test generation rules for each data type.

Chakrabarti and Kumar [26] proposed an approach that aims to generate test cases for testing RESTful Web services. The authors also introduced a prototype tool that supports this approach.

Proposed approaches [23–25] for WSDL-based test data generation tend to generate test cases for testing a single web service operation. Test cases that test a single operation might work for testing most 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.* [9] addressed 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 input data types because of the lack of behavioural information about the service. As a result, many researchers look for other alternative specifications that can provide additional behavioural information, such as contracts and semantic service specifications. For this, Semantic Web Service (SWS) specifications are often used, because they contain more information compared with WSDL. The use of semantic model OWL-S for test data generation is proposed [27–30] not only because of the behavioural information it contains but also because of the semantic information concerning the data types on which the service operates. This semantic information in the form of ontology allows ontology-based test data generation [30].

Ye *et al.* [31] introduced a static BPEL defect analysis system focussing on WSDL-related faults. The authors defined defect patterns related to WSDL elements (partnerLinkType, role, portType, operation, message and property) that help reveal defects related to these elements.

One of the main problems in ScST and testing online systems is the type of test data required [32]. Bozkurt and Harman [32] categorized the test data required by online systems as 'realistic test data' (RTD) and defined it as *data that are both structurally and semantically valid*. The authors discussed the importance of using RTD in ScST and claimed that most of the existing approaches fail to generate RTD or fail to automate the test data generation process.

One of the earliest approaches that aims to address this issue was proposed by Conroy *et al.* The approach by Conroy *et al.* [33] generates test data by using applications with graphical user interfaces (GUIs). This approach harnesses user input data from GUI elements and uses the harnessed data to generate test cases for ScS.

An automated solution was proposed by Bozkurt and Harman [32, 34]. The proposed approach is capable of generating RTD while providing a high level of automation. The authors also presented a framework that supports the approach called ATAM service-oriented test data generator. The approach exploits existing web services as sources of RTD and automatically forms service compositions that are likely to provide the required test data as output. The framework uses data ontologies for composition, and, as a result, it can generate test data for any semantic system. The proposed approach is also capable of generating test data on the basis of user-specified constraints. Bozkurt and Harman [35] also proposed the use of multiobjective optimization and QoS parameters within their approach in order to reduce the cost of test data generation and testing and to increase the reliability of the testing process.

5.2.2. Contract-based test case generation approaches. Design by Contract (DbC) [36] is a software development approach where contracts define the conditions (preconditions) for a component to be accessed and the conditions (postconditions) that need to hold after the execution of methods of that component with the specified preconditions. With the use of 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 [37–40].

DOI: 10.1002/stvr

Because traditional Web services only provide interface information, researchers have proposed contracts for several 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 preconditions and postconditions of operations increases the testability of services.

Heckel and Lochmann [41] proposed the use of the DbC approach for services and discussed 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.* [42, 43] proposed using a technique called test sheets in order to generate unit test cases and test oracles. Test sheets contain contract information that identifies the relation among the operations of a service. The included relations define the effects of each operation from the clients' perspective in order to help validation.

Web Service Description Language extensions proposed to allow WSDL files to accommodate contract information such as preconditions and postconditions. For example, Tsai *et al.* [44] discussed the reasons for an extension to WSDL in order to perform black-box testing and proposed an extended WSDL that carries additional information such as input–output dependency, invocation sequence, hierarchical functional description and sequence specifications. Similarly, Mei and Zhang [45] proposed 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 [41] XML-level contracts also require an extension to WSDL.

Noikajana and Suwannasart [46] proposed the use of a pair-wise testing (PWT) technique to facilitate contract-based test case generation. In the proposed approach preconditions and postconditions are included in Web Service Semantics (WSDL-S) specifications using the Object Constraint Language (OCL).

Askarunisa *et al.* [47] proposed the use of PWT and orthogonal array testing (OAT) in test case generation for semantic services. The authors also compared these two approaches in order to determine which approach performs better in different testing scenarios. The authors used the same contract specifications (OCL and WSDL-S) as Noikajana and Suwannasart.

Liu *et al.* [48] also proposed the use of OCL-based constraint systems. The constraints are included in the SAWDSL semantic service annotations. The proposed approach generates test cases by performing boundary analysis and class division.

Mani *et al.* [49] propose the inclusion of contract information in service stubs. These semantically extended stubs carry contract-like information such as preconditions and postconditions.

Dai *et al.* [28] proposed contracts that can be contained in OWL-S process models. Proposed contracts carry information such as preconditions and postconditions between a service user and a service. Dai *et al.* also presented a framework that is capable of generating test cases and oracles, monitoring test execution and verifying the SUT. Bai *et al.* [27] proposed a testing ontology model that describes test relations, concepts and semantics, which can serve as a contract among test participants.

Saleh *et al.* [50] proposed contracts for data-centric services. The proposed contracts consist of logical assertions and expose data-related business rules. The approach combines DbC and formal methods to prove the correctness of service compositions.

5.2.3. Partition testing approaches. Partition testing is a testing technique that aims to find subsets of the test cases that can adequately test a system. The aim of partition testing is to divide the input domain of the SUT into subdomains, so that selecting or generating a number of test cases from each subdomain will be sufficient for testing the entire domain. In essence, partition testing is much like mutation testing, or sometimes, mutation testing is considered a partition testing technique [51]. In mutation testing, faults are introduced into every subdomain that is known to function correctly in

DOI: 10.1002/stvr

order to measure the effectiveness of the test suite. With the introduction of a fault into a subdomain, it is possible to identify the test cases that belong to that subdomain during test executions.

Heckel and Mariani [52] claimed that Δ-Grammars are more suitable for testing web services than UML diagrams because of their ability to describe the evolution of services. Heckel and Mariani suggested a partition-testing approach based on WSDL definitions and Δ-Grammars. Similarly, Park *et al.* [53] also applied this approach to service selection.

The application of partition testing to Web services is proposed at two different levels. Bertolino *et al.* [20] proposed the use of the category-partition method [51] with XML schemas in order to perform XML-based partition testing. This approach automates the generation of test data by using XML schemas. Bertolino *et al.* introduced a tool that supports this approach called TAXI. Another approach was proposed by Bai *et al.* [27] for OWL-S semantic services. Bai *et al.* introduced a test ontology model that specifies the test concepts and serves as a test contract. The data partitions used by this approach are created using the ontology information.

5.3. Experimental results

This section also follows the classification from the previous section.

5.3.1. Experimental results of specification-based approaches. Bartolini *et al.* [19] ran their experiment on a real service called PICO and performed mutation analysis to observe the effectiveness of their approach. WS-TAXI achieved 72.55% mutation score, and it managed to kill 63% more mutants than a manually generated test suite using an enterprise testing software.

The constraint-based test data generation approach of Ma *et al.* [23] is validated using three versions of a synthetic web service. In this context, synthetic web service means a service implemented to test the testing approach. During the experiments, 4096 different test data was generated for a complex data type and revealed two bugs in two different services.

Bai *et al.* [9] experimented on 356 real services with 2050 operations in total. For the 2050 operations, they generated 8200 test cases to cover all operations and valid messages. Test cases were generated to perform two analysis, constraint analysis and boundary analysis, on data types and operation dependency analysis. According to the results by the authors, 7500 of the generated test cases were exercised successfully on 1800 operations. The authors' explanation for the unsuccessful test cases is the mistakes in WSDL documents.

Conroy *et al.* [33] experimented on the web services of two real applications: Accenture People Directory and University Data. The authors compared the effort it takes to write a parser that extracts information from data files with their approach. According to the authors, it took 2 h to write a parser and 10 min for their approach to generate test cases.

Bozkurt and Harman [32] experimented on 17 existing commercial web services in two different case studies. Testing is performed on four of these services, whereas the rest are used in the test data generation process. The authors compared their approach against random test data generation (state-of-the-art automated test data generation method for ScST). In generating structurally valid test data, the random test data generation method achieved 8% success rate in case study 1 and 24% success rate in case study 2, whereas the proposed approach achieved 94% and 100% success rates in the same case studies. In generating semantically valid data, random testing achieved 0% and 34% success rates, whereas the proposed approach achieved 99% and 100% success rates. The authors also evaluated the approach's ability to generate test data on the basis of tester constraints, and the approach achieved 100% success rate for all of the given constraints in both of the case studies.

5.3.2. Experimental results of contract-based approaches. Mei and Zhang [45] experimented on two synthetic web services: triType and mMiddle. The test suite created by this approach achieved 95% mutation score and 96% and 81.5% reduction in test cases while maintaining the test coverage.

Askarunisa *et al.* [47] experimented on two real and two synthetic web services. In their experiments, PWT managed to reveal 18 faults, whereas OAT only revealed 13 faults. In terms of test case reduction performance, PWT and OAT were compared with random test data generation. Both PWT

DOI: 10.1002/stvr

and OAT have shown significant ability to reduce the number of required test cases (up to 99.76% reduction by using PWT and 99.84% reduction by using OAT).

Mani *et al.* [49] experimented on a set of services from an IBM project, and their approach achieved 37% reduction in test suite size.

Noikajana and Suwannasart [46] experimented on two example services: RectangleType and IncreaseDate. The authors compared the effectiveness of their test cases generation approach against test cases generated using a decision table by performing mutation testing. The test suit generated by this approach achieved 63% (RectangleType) and 100% (IncreaseDate) mutation score, whereas test suit from decision table achieved 85% and 56% mutation score. This approach also outperformed the decision table-based approach in multiple condition coverage (achieved 100% coverage).

5.3.3. Experimental results of partition testing approaches. According to the experiments of Bai *et al.*, with the use of partitioning, a 76% reduction is achieved, reducing 1413 randomly generated tests for 17 partitions to 344. Comparing the partition technique of Bai *et al.* 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 the 20 test cases that were selected using partitioning achieved the same coverage.

5.4. Discussion

The effort for specification-based test case generation for WST is divided into two categories: generating valid test data with the aim of revealing faults and generating invalid test data with the aim of measuring the robustness of services. The approaches aiming to test the robustness services are discusses in detail in Section 7.

Specification-based test data generation approaches focus on generating test data to perform boundary analysis or constraint-based tests. These tests can be very valuable and useful to all the SOA stakeholders. Most of the publications highlight the fact that they generate high numbers of test cases with minimal effort, thereby supporting the claim that they help reduce the manual labour of generating test cases.

A possible disadvantage of the test case generation approaches described previously is type of test data they are able to generate. Almost all the approaches use XML data type constraints and other constraints provided by either the tester or the provider and generate test cases by using these constraints. However, none of them aim to generate RTD except Conroy *et al.* [33] and Bozkurt and Harman [32].

Unfortunately, running a large number of test cases might be a problem in WST because of cost of invoking services or access limitations. For example, for the developer and the provider where testing cost is minimal, these approaches can be very useful. However, for the integrator and the certifier, running all the test cases generated by these approaches can be very expensive. This signifies the importance of test case selection/reduction techniques in WST.

The results from the experiments indicate that partition testing can help with selection. Selection approaches can be very effective when combined with an automated data generation method such as the approaches from Section 5.2.1. Partition testing can be very useful to the integrator and the certifier where cost of testing is high.

Contract-based WST can achieve better and more efficient testing than WSDL-based testing because of increased testability. One of the most important benefits of contracts, as was highlighted by the work in this section, is that they help with test case selection/reduction by enabling the application of test case generation techniques such as PWT and OAT. The experimental results provide evidence to the effectiveness of test case reduction using contracts. On the other hand, for the developer and the provider, contracts increase the cost of service creation. Regardless of the benefits that contracts provide, the cost of creating them makes DbC less widely practised. A similar problem is also faced by SWS where creating semantic specifications is laborious.

Currently, there is no DbC standard for SOA. Some of the approaches in this section [28, 41, 44, 45] propose extensions to standards such as WSDL and OWL-S in order to solve this

DOI: 10.1002/stvr

problem. Many existing SWS proposals already include contract-like information such as preconditions and postconditions and their effects on execution. However, further additions to SWS may be required in order to improve testability.

6. UNIT TESTING OF SERVICE-CENTRIC SYSTEMS

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 to be units. In terms of web services, the operations provided by a service can be considered as units to be tested. A service composition or choreography may also be considered as a service unit.

Service-centric unit testing (SCUT) is generally performed by sending and receiving SOAP or Hypertext Transfer Protocol (HTTP) messages. The tester generates the SOAP/HTTP messages for the operation/application under test by using the information from the WSDL file. In this way, unit testing can be used to verify both the correctness of the WSDL and the correct functioning of the SUT.

There are industrial tools that provide some level of automation to SCUT such as Parasoft SOAtest [54], SOAP Sonar [55], HP Service Test [56] and the Oracle Application Testing Suite [57]. Even though these tools help reduce the manual labour 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/HTTP requests for each test case. In some of these tools, even verification of test results has to be performed manually such as in SOAtest. Automated SCUT remains at a similarly immature and consequently labour-intensive state as more general test automation.

6.1. Perspectives in unit testing

Service-centric unit testing can be performed in a white-box manner as well as a black-box manner depending on the access to the service implementation. In SOA, the developer is the only stakeholder who can perform structural tests. Unit testing at the service level using specifications (including contracts) is commonly performed by stakeholders other than the developer.

Unit testing of stateless Web services might be performed differently than stateful services. While testing stateless services, each operation of a service can be accepted as a unit. As a result, the integrator or the certifier tests the necessary operations separately for these services. However, for stateful services, operations can be tested together and, for some tests, it has to be so-tested to capture and test stateful behaviour. For such services, the developer has the ability to manipulate the service state during testing.

Unit testing of service compositions can be performed in two different ways: real-world testing and simulation. Real-world testing of service compositions can be performed by the integrator using existing services. The integrator may also perform simulations by using stub or mock services to test the business process.

6.2. Unit testing approaches

The need for tools that can automate unit testing has been addressed by the research community. For example, Sneed and Huang [25] introduced 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 preconditions 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 the necessary assertions.

Lenz *et al.* [58] proposed a model-driven testing framework that can perform unit tests. In this approach, JUnit tests are generated using requirement specifications and platform-independent test specifications, based on the UML 2 Testing Platform. Both of these required specifications are provided by the service provider.

DOI: 10.1002/stvr

Zhang *et al.* [59] presented a framework based on Haskell modules that is capable of generating and executing test cases from WSDL specifications. The HUnit component organizes the test plans and provides unit test execution.

One of the main problems of software testing is the oracle problem [60, 61]. 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 associated with each test input. In ScST, the tester often does not have any reliable test oracles available to support testing. The lack of a test oracle is one of the challenges of automated ScST.

Test oracle generation was addressed by Chan *et al.* [62]. Chan *et al.* proposed a metamorphic testing framework that is capable of performing unit testing. Metamorphic testing [63] can potentially solve the test oracle problem by using metamorphic relations. These relations are defined by the tester for each test suit. Chan *et al.* also proposed the use of metamorphic services that encapsulate a service and imitates its functionality. Verification for the SUT is provided by the encapsulating metamorphic service that verifies the input and the output messages against the metamorphic relations. The framework of Chen *et al.* enables service users to determine the expected results but requires the use of relations that can be costly for the provider.

Similarly, Heckel and Lochmann [41] generated test oracles by using pregenerated contracts. The contracts, created using the DbC approach, are supplied by the provider and carry information such as preconditions and postconditions.

Atkinson *et al.* [43] proposed the use of a technique called *test sheets* in order to generate unit test cases and test oracles. The test sheets approach uses tables that define test cases similar to the Framework for Integrated Test [64], 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 included 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 was proposed by Tsai *et al.* [65]. Tsai *et al.* proposed the adaptation of blood group testing to Web services and called this technique Adaptive Service Testing and Ranking with Automated Oracle Generation and Test Case Ranking (ASTRAR) [10, 66]. ASTRAR is similar to n-version testing where multiple 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 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.

Yue *et al.* [67, 68] proposed a message-based debugging model for Web services. The authors presented an operational model and a context inspection method for message-based debugging. The proposed approach is able to trace service behaviours, dump debugging information and manage states and behavioural breakpoints of debugged services.

Unit testing of service compositions using BPEL has also been addressed in the literature. According to Mayer and Lübke [69], BPEL unit testing is performed in two ways: simulated testing and real-world testing. In simulated testing, as opposed to real-world testing, BPEL processes are run on an engine and contacted through a test API. This mechanism replaces regular deployment and invocation. In BPEL testing, service stubs or mocks can be used instead of the services that participate in the process. Mayer and Lübke [69] proposed a framework that is capable of performing real-world unit testing. This framework can replace participating services with service mocks. The framework also provides a mechanism for asynchronous messaging by providing an implementation of WS-Addressing [70]. Li *et al.* [71] adopted a different approach for unit testing in which BPEL processes are represented as a composition model. Similar to Mayer and Lübke's framework, the framework of Li *et al.* uses stub processes to simulate the parts that are under development or inaccessible during the testing process.

Mani *et al.* [49] proposed the idea of semantic stubs. Semantic stubs carry additional information such as preconditions and postconditions. Semantic stubs enable input message verification, expected output generation and exception message generation for the simulated services. The

DOI: 10.1002/stvr

authors also presented a framework that automatically generates stubs from semantically annotated WSDL descriptions.

Palomo-Duarte *et al.* [72] introduced a tool that dynamically generates invariants for BPEL processes. The proposed invariants reflect the internal logic of BPEL processes and are generated from the execution of tester-provided test cases. Takuan assists in discovering bugs and missing test cases in a test suite.

Ilieva *et al.* [73] introduced a tool for end-to-end testing of BPEL processes called TASSA. The TASSA framework is built according to SOA principles and is a platform-independent and composable system. The tool provides simulated testing and offers an injection tool, a data dependency analysis tool, a test case generation tool and a value generation tool. Reza and Van Gilst [74] introduced a framework that is aimed at simulating RESTful Web services. Li *et al.* [75] introduced a toolkit called SOArMetrics for evaluating SOA middleware performance and application testing.

6.3. Experimental results

Sneed and Huang [25] experimented on an eGovernment project with nine services. They generated 22 requests per service, and 19 out of 47 verified responses contained errors. They have revealed 450 total errors in the whole project, which 25 of them caused by the services.

Tsai *et al.* [10] experimented on 60 different versions of a synthetic real-time service. The approach achieved a 98% probability of establishing a correct oracle and a 75% probability in ranking the test cases correctly according to their potency. The authors claimed that the high scores in these parameters should lead to a better service ranking by detecting faulty services faster within the process.

Mani *et al.* [49] experimented on a set of services from an IBM project. The authors evaluated two different abilities of their approach test suite reduction and execution efficiency. The approach achieved a 37% reduction in test cases while maintaining the effectiveness. The results also provide evidence to support the authors' claim that semantic stubs provide faster executions compared with remote services.

6.4. Discussion

Unit testing is one of the most important testing techniques that every system must undergo. The main challenge faced in SCUT is the high cost of testing due to manual test case generation and test execution. This cost 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, although they lack automated test oracle generation.

Tsai *et al.* [65], Chen *et al.* [76] and Heckel and Lochmann [41] addressed the oracle problem. The approach of Tsai *et al.* provides fully automated test oracle generation without the need of any extra information from the provider. However, the approach needs to discover services with similar business logic, and it must meet any costs associated with the use of other services.

At the composition level, increased cost of testing due to invoked services can be reduced by introducing service stubs. Approaches such as those of Mayer and Lübke [69], Mani *et al.* [49], Van Gilst [74], Ilieva *et al.* [73] and Li *et al.* [71] address this issue by providing functionality for generating and using stubs. Using stubs also help in reducing the cost of invoking services that perform business transactions. Unfortunately, using stubs does not overcome the need to test with real services (run-time testing). As a result, this problem still remains as an open problem and needs more effective solutions.

7. FAULT-BASED TESTING OF SERVICE-CENTRIC SYSTEMS

According to Morell [77], 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 seek to prove the nonexistence of known faults rather than trying to find unknown faults that do exist.

DOI: 10.1002/stvr

In SOA, fault-based testing is extremely important for the stakeholders who have to test services for their robustness. The provider and the certifier must test services by using fault-based approaches during reliability measurement. Fault-based testing can also help the integrator to observe how service composition and individual services behave in unexpected conditions.

Hanna and Munro [21] classified test data generation for different testing techniques and also surveyed fault-based testing research in the Web services domain. These testing techniques are the following:

1. *interface propagation analysis*, which 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; and
4. *equivalence partitioning with invalid partition class*, where the input space or domain is partitioned into a finite number of equivalent classes with invalid data.

In the present survey, the research undertaken in fault-based ScST is categorized according to the level that faults are generated. Fault-based testing of ScS 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.

7.1. Perspectives in fault-based testing

Fault-based testing can be useful to all the stakeholders of SOA, but each stakeholder performs it in a different manner according to their level of observability and control. For example, fault-based testing using SOAP and XML perturbation, where messages among services are captured, can only be performed by the integrator, whereas the other approaches where faulty messages are generated from service specifications can be performed by the remaining stakeholders in SOA. In mutation testing, the developer has also the advantage of being able to perform standard mutation analysis by introducing faults into the workflow. The integrator has a similar advantage when performing mutation testing on compositions. On the other hand, the provider and the certifier can only perform specification mutation on service specifications.

One important problem in fault-based testing is its cost. Performing fault-based testing can be costly especially at the integrator and the certifier side. For example, approaches using mutation testing can increase the cost greatly because they require generation of many mutants and running each mutant with many test cases to kill it.

7.2. Fault-based testing approaches

The approaches in this section are divided into three categories on the basis of the abstraction level at which faults are injected.

7.2.1. XML/Simple Object Access Protocol perturbation. XML/SOAP perturbations are performed by using faulty SOAP messages. Faulty messages are generated from the captured messages (between services or a user) by injecting faults before sending them or just by sending a faulty SOAP message to the service. After perturbations, the service's behaviour with the faulty message is observed for verification.

One of the earliest examples of SOAP perturbation was proposed by Offutt and Xu [24]. Offutt and Xu proposed three different types of perturbations:

1. *Data value perturbation*, which is performed by modifying the values in a SOAP message.
2. *Remote procedure calls communication perturbations*, which is performed by modifying the arguments of the remote procedures. Offutt and Xu proposed the application of mutation analysis to syntactic objects and data perturbation to SQL code. SQL code perturbation also facilitates SQL injection testing.

DOI: 10.1002/stvr

3. *Data communication perturbations*, which is used for testing messages that include database relationships and constraints.

Xu *et al.* [78] proposed an approach where perturbation is applied to XML schemas in order to generate test cases. Xu *et al.* defined XML schema perturbation operators for creating invalid XML messages by inserting or deleting data fields. Almedia and Vergilio [79] also adopted the same approach and proposed a tool called SMAT-WS that automates the testing process. Almedia and Vergilio also introduced some new perturbation operators for XML perturbation. Hanna and Munro [21] tested robustness of services by violating the input parameter specifications from WSDL files. Hanna and Munro's approach can test both the service itself and the platform the service resides in. Zhang and Zhang [80] proposed boundary value fault injection testing in order to help with selecting the reliable services. Similarly, Vieira *et al.* [81] proposed a framework that applies fault injection to the captured SOAP messages. Martin *et al.* [82] proposed a framework called WebSob that tests services for robustness. WebSob tests service methods with extreme or special parameters.

Salva and Rabhi [83] proposed an approach aimed at testing the robustness of stateful Web services. The authors performed an analysis on SOAP service observability in order to distinguish between the types of perturbations that generate SOAP faults at the SOAP processor level and at the service level. The approach uses the only two perturbation methods that are handled at the service level: SOAP value perturbations and operation name perturbations. According to the authors, the other proposed perturbation methods on parameter types such as deleting, adding and inverting are handled at the SOAP processor level and thus does not test the service itself.

Tsai *et al.* addressed the problems of test data ranking and fault-based testing within a single approach. Tsai *et al.* [29] proposed 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.

Wang *et al.* [84] proposed a fault injection method for BPEL processes using service stubs. The proposed stubs can generate business semantics-oriented faults by mimicking unexpected behaviours of real services. The stubs are generated automatically from WSDL definitions, but the code that causes the faults needs to be implemented and inserted manually.

7.2.2. Network-level fault injection. Network-level fault injection is a fault injection approach in which faults are injected by corrupting, dropping and reordering the network packages. Looker *et al.* [85] proposed the use of this technique along with a framework called the Web Service Fault Injection Tool. At the network level, latency injection can be performed along with SOAP perturbation. Web Service Fault Injection Tool can perform both SOAP perturbations and latency injections. Looker *et al.* also proposed 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 proposed 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).

Juszczyk and Dustdar [86, 87] introduced a SOA test bed based on the Genesis2 framework for testing ScS. The test bed is capable of simulating QoS of the participating services and also of generating issues such as packet loss and delay and service availability.

7.2.3. Mutation of Web service specifications. One of the first examples of web service mutation testing (WSMT) was applied to WSDL specifications. Siblini and Mansour [88] proposed the use of WSDL mutation for detecting interface errors in web services.

Mei and Zhang [45] defined mutation operators for contracts. The contracts in this approach are included in the extended WSDL specifications that were proposed by the authors.

The next step in WSMT was to take the mutation into the SWS. The amount of information provided by OWL-S allows for the application of mutation operators at different levels compared with WSDL mutation. For example, Lee *et al.* [89] proposed an ontology-based mutation to measure semantic test adequacy for composite services and for semantic fault detection.

DOI: 10.1002/stvr

Similarly, Wang and Huang [90, 91] proposed another ontology-based mutation testing approach that uses OWL-S requirement model. Wang and Huang suggested a modified version of the requirement model enforced with Semantic Web Rule Language [92] 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.

Apilli [93] and Watkins [94] proposed the use of combinatorial testing approaches for fault-based testing of Web services. The proposed approaches focus on known faults in order to avoid possible combinatorial explosion. The reduction in combinations is achieved by restricting input conditions.

The fault injection approach of Fu *et al.* [95] differs from other work 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.

As for all the testing methodologies, automation is important for fault-based testing. Several tools for fault-based ScST are also introduced. Laranjeiro *et al.* [96, 97] presented a public web service robustness assessment tool called wsrbench [98]. wsrbench provides an interface for sending SOAP messages with invalid service call parameters. wsrbench is also capable of providing the tester with detailed test results for a service. Laranjeiro *et al.* [99] also proposed the use of text classification algorithms to automate the classification of the robustness test results.

Bessayah *et al.* [100] presented a fault injection tool for SOAP services called WSInject. WSInject is capable of testing both atomic services and service compositions and combining several faults into a single injection.

Domínguez-Jiménez *et al.* [101] introduced an automated mutant generator for BPEL processes called GAmera. GAmera uses a genetic algorithm (GA) to minimize the number of mutants without losing relevant information. GAmera is also able to detect potentially equivalent mutants.

7.3. Experimental results and discussion

Fault-based ScST at the service level can be very effective when the tester wants to check for common errors such as interface errors, semantic errors and errors that can be caused by the Web services platform. Similar to the boundary and constraint-based analyses, the results from the experiments show that fault-based testing can reveal more faults than positive test cases.

The results of SOAP perturbations prove the effectiveness of this approach in the rate of faults revealed during experiments. For example, during Offutt and Xu's experiments, 18 faults were inserted into the Mars Robot Communication System and 100 data value perturbation, 15 remote procedure calls communication perturbations and 27 data communication perturbations tests were 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 Mars Robot Communication System and additionally on the supply chain management application from Web Services Interoperability Organization (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, seven of these services showed no robustness problems. Vieira *et al.* also highlighted that a significant number of the revealed errors are related to database accesses. Tsai *et al.* experimented on 60 best buy stock web services with 32 test cases and in these experiments negative test cases revealed more faults than positive test cases. Salva and Rabhi [83] experimented on the Amazon E-Commerce Service. 30% of the test cases generated using their approach caused unexpected results.

Looker *et al.* [85] experimented on a simulated stock market trading system that contains three 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.

Domínguez-Jiménez *et al.* [101] experimented on the loan approval example. The authors tested GAmera with different configurations and mutant sizes ranging from 50% to 90% of the possible

DOI: 10.1002/stvr

22 mutants in order to discover the optimal subset of mutants. GAmera was able to reduce the size of the mutants for all subsets without losing relevant information. However, each mutant population included two equivalent mutants. In each population, between two and three of the generated mutants were not killed by the test suite.

Bessayah *et al.* [100] experimented on the travel reservation service (TRS is an example BPEL process) composed of three services. During the experiments, WSInject was able to reveal several interface errors by applying SOAP perturbation (invalid data) and two communication faults by applying network-level fault injection (SOAP message delay).

Laranjero *et al.* [99] experimented on 250 existing web services to evaluate their automated robustness test result classification method. The authors suggested the use of five text classification algorithms: Hyperpipes, Ibk, large linear classification, Naïve Bayes and symbolic model verifier. The algorithms successfully classified 96.63%, 98.89%, 98.75%, 90.31% and 96.55% of the detected robustness problems, respectively.

According to the results of the proposed approaches, mutation testing is effective for measuring test case adequacy in web services. Mei and Zhang's [45] WSDL mutation achieved 95%, Wang and Huang's [90] OWL-S mutation achieved 98.7% and the OWL-S mutation of Lee *et al.* [89] 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. The approach of Lee *et al.* also helped with equivalent mutant detection by detecting 25 equal mutants out of 686 generated mutants.

Because 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. The information on the behaviour of a service under an unexpected situation is valuable to the integrator to implement a robust service composition.

Fu *et al.* experimented on four Java web services: the FTPD ftp server, the JNFS server application, the Haboob Web server and the Muffin proxy server. The approach achieved over 80% coverage on fault handling code during experiments. The approach by Fu *et al.* can only be applied to Java services and performed by the developer. This approach does not guarantee the execution of the error recovery code in the case of an error nor the correctness of the recovery action.

8. MODEL-BASED TESTING AND VERIFICATION OF SERVICE-CENTRIC SYSTEMS

Model-based testing is a testing technique where test cases are generated using a model that describes the behaviour of the SUT. Advantages of MBT, 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 [102].

Formal verification of service compositions is popular because of formal verification methods' ability to investigate behavioural properties. The earliest work on formal verification of service compositions dates back to 2002. The existing work that has been undertaken in formal verification of service compositions was compared by Yang *et al.* [103]. Morimoto [104] 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 until 2009 in MBT (including formal verification) of ScS according to the testing technique used.

8.1. Perspectives in model-based testing

Perspectives in MBT of ScS are based on the source from which test models are created. The MBT approaches where models are created from service specifications, such as OWL-S or WSDL-S, and other where models are created from service composition languages, such as BPEL, can only be performed by the integrator.

DOI: 10.1002/stvr

8.2. Model-based testing approaches

This section is divided into four groups on the basis of the MBT method used.

8.2.1. Model-based test case generation. The application of MBT methods to ScS has been widely proposed, and the application of different MBT and verification methods such as symbolic execution and model checking are also proposed. Many of these proposed approaches are capable of generating test cases.

The use of graph search algorithm(s) (GSA) and path analysis (PA) (using constraint solving) is the earliest proposed MBT method for ScST. The generation of control flow graph(s) (CFG) from BPEL processes is widely adopted [105–107]. In these approaches, test cases are generated from test paths that are created by applying GSA to the CFG of the process. The difference between GSA and PA is the way that test data is generated. In PA, test data for each path is generated using constraint solvers, whereas in GSA, the algorithm itself generates the test data.

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.* [106] proposed 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.* [107] proposed 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.

Lallai *et al.* [108, 109] proposed the use of an automaton 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 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.* used a tool called BPEL2IF, and for test generation another tool called TESTGen-IF is used. TESTGen-IF is capable of generating test cases for the IF Language. The IF and WS-TEFSM can both model event and fault handlers and termination of BPEL process. Similarly, Cao *et al.* [110, 111] proposed the use of TEFSM for BPEL processes. The authors also introduced a tool that automates the proposed approach called Web Services, Online Testing Framework (WSOTF).

Endo *et al.* [105] proposed the use of the CFG approach in order to provide a coverage measure for the existing test sets. Endo *et al.* proposed a new graph called the parallel control flow graph that contains multiple CFG representing a service composition and communications among these CFG. They also presented a tool that supports the proposed technique called ValiBPEL.

Li and Chou [112] proposed the application of combinatorial testing to stateful services. The authors proposed a combinatorial approach that generates multisession test sequences from single-session sequences using multiplexing. The proposed approach generates condition transition graphs, which are used with a random walk algorithm. The authors [113] also proposed an abstract guarded finite state machine (FSM) to address testability issues in conformance testing.

Belli and Linschulte [114] proposed an MBT approach for testing stateful services. The authors introduced a model that captures events with the corresponding request and response for each event called event sequence graph (ESG). In the proposed approach, ESGs are supported by contract information in the form of decision tables. The authors also introduced a tool that generates test cases from decision tables. Endo *et al.* [115] also proposed the use of ESG in order to perform a grey-box testing, focussing on code coverage.

Hou *et al.* [116] addressed the issues of message-sequence generation for BPEL compositions. In this approach, BPEL processes are modelled as a message sequence graph and test cases are generated using this message sequence graph.

Paradkar *et al.* [117] proposed a model-based test case generation for SWS. In this approach, test cases are generated using predefined fault models and input, output, preconditions and effects information from semantic specification.

Guangquan *et al.* [118] proposed the use of UML 2.0 activity diagram [119], a model used for modelling workflows in systems, to model BPEL processes. After modelling the BPEL process, a

DOI: 10.1002/stvr

depth-first search method combined with the test coverage criteria is performed on the model in order to generate test cases.

Ma *et al.* [120] proposed the application of Stream X-machine based testing techniques to BPEL. A stream X-machine [121] is an extended FSM (EFSM) that includes design-for-test properties. It can be used to model the data and the control of a system and to generate test cases.

Casado *et al.* [122] claimed that there is no practical approach to test long-lived web service transactions and proposed an approach that aims to generate test cases for testing web service transactions. The authors introduced a notation and system properties for testing the behaviour of service transactions that comply with Web Services Coordination (WC-COOR) and Web Services Business Activity (WS-BA). In this approach, test cases are generated using fault tree diagrams.

Search-based test data generation has attracted much recent attention [123–125]. Search-based test data generation techniques enable automation of the test generation process, thus reducing the cost of testing. Blanco *et al.* [126] proposed the use of scatter search, a metaheuristic technique, to generate test cases for BPEL compositions. 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.

Tarhini *et al.* [127, 128] proposed two new models for describing composite services. In this approach, the SUT is represented by two abstract models: the task precedence graph (TPG) and the timed labelled transition system (TLTS). The TPG models the interaction between services, and the TLTS models the internal behaviour of the participating services. Tarhini *et al.* proposed three different sets of test case generation for testing different levels of service composition. Test cases in the first set aim to perform boundary value testing by using the specifications derived from the WSDL file. The second set is used for testing the behaviour of a selected service. The third set tests the interaction between all the participating services, and test cases for this set are generated using TPG and TLTS.

8.2.2. Model-based testing and verification using symbolic execution. Symbolic execution is used as a basis for a verification technique that lies between formal and informal verification according to King [129]. 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 by using symbolic execution is the BZ-Testing-Tools (BZ-TT) method [130]. 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. It also performs 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 services by using symbolic execution was proposed by Sinha and Paradkar [131]. Sinha and Paradkar proposed 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 proposed the use of four different test data generation techniques: full predicate coverage, mutation-based, projection coverage and the BZ-TT method. For full predicate coverage, each condition of the EFSM is transformed into disjunctive normal form [132], and test sequences covering each transition are generated. For 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 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.* [133] proposed another approach that uses symbolic execution in order to test service compositions. In this approach, a service composition is first translated into a symbolic transition system (STS); then, a symbolic execution tree is created using STS of the composition. The approach of Bentakouk *et al.* takes coverage criteria from the tester and generates the set of execution paths on symbolic execution tree. These generated paths are executed using a test oracle

DOI: 10.1002/stvr

over the service composition. Bentakouk *et al.* claimed that using symbolic execution helps avoid state explosion, overapproximation and unimplementable test case problems that are caused by labelled transition systems. As a result, the approach of Bentakouk *et al.* can handle rich XML-based data types.

Zhou *et al.* [134] proposed an approach aimed to test service compositions by using Web Service Choreography Description Language. The approach uses a dynamic symbolic execution method to generate test inputs. In this approach, assertions are used as test oracles. The approach uses an satisfiability modulo theory solver for generating inputs that satisfy path conditions.

Escobedo *et al.* [135] proposed an approach for testing BPEL processes by using a specialized labelled transition system called IOSTS. The approach is capable of performing both real-world testing and simulated testing. One important aspect of the approach is that it assumes that Web service specifications are not available. If the behaviour of the services can be simulated, the approach performs simulated model-based unit testing. In real-world testing, the approach is also capable of coping with problems of service observability.

8.2.3. Model-based testing and verification using model checking. Model checking is a formal verification method and is described as a technique for verifying finite state concurrent systems by Clarke *et al.* [136]. Model checking verifies whether the system model can satisfy the given properties in the form of temporal logic. During the proofing process, the model checker detects witnesses and counterexamples for the properties of interest. 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.

Automata, either finite state automata [137] or FSMs [138], are often used to represent finite state systems. In BPEL testing, automata are often the target of transformation. BPEL processes are first transformed into automata models; these models can subsequently be transformed into the input languages of model checking tools such as Simple Promela Interpreter (SPIN) [139], NuSMV [140] or Berkeley Lazy Abstraction Software Verification Tool (BLAST) [141].

Fu *et al.* [142] proposed 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 [143]) 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 [144] with bounded queues (directly) or with synchronous communication on the basis of the result of the synchronizability analysis. The SPIN tool takes Promela as the input language and verifies its linear temporal logic (LTL) [145] properties. Interactions of the peers (participating individual services) of a composite web service are modelled as conversations, and LTL is used for expressing the properties of these conversations.

García-Fanjul *et al.* [146] used a similar method to that of Fu *et al.* [142] in order to generate test cases. The approach of García-Fanjul *et al.* differs from the approach of Fu *et al.* in transforming BPEL directly to Promela. The approach of García-Fanjul *et al.* generates test cases by using the test case specifications created from counterexamples that 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 suite 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.* [147] proposed another test case generation method using model checking for 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 proposed an automaton called Web service automaton [148] that is used to transform BPEL into Promela for SPIN or symbolic model verifier (SMV) for NuSMV model checker. Web service automaton aims to include BPEL data dependencies that cannot be represented in other automata-based formalisms. This approach generates test cases by using counterexamples to perform conformance tests on BPEL and by using WSDL to test web service operations.

DOI: 10.1002/stvr

Huang *et al.* [149] proposed the application of model checking to SWS compositions using OWL-S. The proposed approach converts OWL-S specifications into a C-like language and the Planning Domain Definition Language for use with the BLAST model checker. With the use of BLAST, negative and positive test cases can also be generated. Huang *et al.* proposed an extension to the OWL-S specifications and the Planning Domain Definition Language in order to support this approach and used a modified version of the BLAST tool.

Jokhio *et al.* [150] proposed the application of model checking to the WSMO goal specifications. The approach translates the goal specifications to B abstract state machine, which is used to generate test cases by using the assertion violation property of the ProB model checker.

Qi *et al.* [151] claimed that existing software model checkers cannot verify liveness in real code and proposed an approach that aims to find safety and liveliness violations in ScS. The approach uses the Finite Trace LTL Model Checking tool to determine whether an SUT satisfies a set of safety and liveness properties. The authors also claimed that this is the first approach for C++ web services.

Betin-Can and Bultan [152] proposed 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 modelled using a hierarchical state machine (HSM) model. Betin-Can and Bultan proposed a modular verification approach using Java PathFinder (JPF) [153], a Java model checker, to perform interface verification, SPIN for behaviour verification and synchronizability analysis.

Ramsokul and Sowmya [154] proposed the modelling and verification of Web service protocols via model checking. Ramsokul and Sowmya proposed a distributed modelling language based on the novel Asynchronous Extended Hierarchical Automata (ASEHA), which is designed for modelling functional aspects of the service protocols. ASEHA model of service protocols are translated into Promela, and correctness of the protocol is verified by the SPIN tool.

Guermouche and Godart [155] proposed a model checking approach for verifying service interoperability. The approach uses the UPPAAL model checker and includes timed properties in order to check for timed conflicts among services. The approach is capable of handling asynchronous timed communications.

Yuan *et al.* [156] proposed an approach for verifying multibusiness interactions. In the approach, business processes are formalized as pi-calculus expressions, which are then translated into SMV input code in order to use with SMV model checker.

8.2.4. Model-based testing and verification using Petri nets. Petri nets are widely used for specifying and analysing concurrent, asynchronous, distributed, parallel, nondeterministic and/or stochastic systems [157]. Petri nets allow different analyses on the model such as reachability, boundedness, deadlock, liveness, reversibility, fairness and conservation analyses. Petri nets can also be used for measuring test case coverage.

Petri nets have also been used in MBT of Web services. For example, Dong and Yu [158] proposed a Petri net-based testing approach where high-level Petri nets (HPNs) 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 data types and policy-based constraints specified by the tester provide the necessary constraints for test data generation.

Wang *et al.* [30] proposed the generation of test cases using Petri nets and ontology reasoning. In this approach, Petri net models that are used to describe 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.* [159], who proposed a method for transforming BPEL processes into Petri nets with formal analysis of BPEL processes using Petri net models. Two tools are used to automate the transformation and analysis: BPEL2PNML [160] and WofBPEL [160]. 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.

DOI: 10.1002/stvr

Schlingloff *et al.* [161] proposed a Petri net-based model checking approach using the LoLA model checking tool [162] to verify BPEL processes. Schlingloff *et al.* introduced a usability analysis to verify the expected behaviour of participating services.

Lohmann *et al.* [163] addressed the problem of analysing the interaction between BPEL processes by using a special class of Petri nets called open workflow net (oWFN). Lohmann *et al.* introduced two tools that support this approach called BPEL2oWFN, which transforms BPEL to oWFN or Petri net. oWFN model is used by another tool called Fiona that analyses the interaction behaviour of oWFNs. Petri net models are used with model checkers to verify the internal behaviour of a BPEL process.

Moser *et al.* [164] proposed a method that increases the precision of Petri net-based verification techniques. The authors claimed that most of the existing verification methods neglect data aspects of BPEL processes. The authors highlighted that their approach incorporates data dependencies into analysis. Data flow is extracted from BPEL by creating a concurrent single static assignment form. Important data items in concurrent single static assignment form are identified and mapped into the Petri net model.

Different flavours of Petri nets are also used in modelling BPEL because of their more expressive nature. For example, the approach of Yang *et al.* [103] is one of the earliest works that proposes the use of coloured Petri nets (CP-Nets) [165], an extended Petri net formalism, for modelling BPEL processes. Yang *et al.* listed 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 [166] for CP-Nets analysis and can also verify the BPEL to CP-Nets transformation.

Yi *et al.* [167] also proposed a BPEL verification framework that uses CP-Nets. Yi *et al.* claimed that CP-Nets are more expressive than FSM and Petri nets and proposed 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.* [168] proposed the use of HPNs. The proposed approach uses a modified version of a tool called Poses++, which was 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.* [169] proposed the use of timed predicate Petri nets 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.* claimed that these annotations can be used in verification of nonfunctional properties of BPEL as well. This is supported by a tool called MCT4WS that allows automated verifications for service compositions.

Xu *et al.* [170] proposed the use of the synchronized net model, a model based on Petri nets. For this approach, Xu *et al.* used a transformation tool called BPEL2PNML, capable of transforming BPEL into Petri Net Markup Language (PNML) [171]. 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 MBT and verification. For example, Li *et al.* [172] proposed a model-based approach for test coverage analysis. The proposed approach uses Resource Description Framework [173], a language for representing information about resources in the World Wide Web, 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.

Yang *et al.* [174] proposed an approach that increases the effectiveness of static defect detection for BPEL processes. The approach uses a CFG representation of BPEL processes in order to perform static analysis. The authors claimed that incorporating the effects of the BPEL's dead path elimination into static defect analysis reduces both false positives and negatives.

Felderer *et al.* [175, 176] proposed a model-driven testing framework called Telling Test Stories (TTS). TTS enables test-driven requirements and testing for ScS. Felderer *et al.* introduced a domain-specific language that allows formalization of system requirements and test model and test cases to be specified on the basis of the concepts of the requirements. Two models are defined in the TTS framework: the system model that describes system requirements at business level and the test

DOI: 10.1002/stvr

model that contains test case specifications. The TTS framework can also ensure the quality of the test artefacts.

Frantzen *et al.* [177] proposed a model-driven development and testing platform called PLASTIC [178]. PLASTIC is supported by two tools that help MBT called Jambition and Minerva. These two tools provide support for offline functional validation for services. In this approach, an STS representation of services is provided by the provider. Jambition can automatically test services by using the provided STS. The authors claimed that Jambition is the only tool that can provide automated on-the-fly testing.

8.3. Experimental results

Huang *et al.* [149] experimented on a synthetic online shopping system. The authors generated seven positive test cases using BLAST and nine negative test cases using the so-called Swiss cheese approach [29] with 100% pass rate.

Jokhio *et al.* [150] experimented on the WSMO example of the Amazon E-Commerce service. The authors were able to generate test cases from two types of trap properties: boundary coverage (three test cases) and modified condition/decision coverage (30 test cases).

The automated conformance test generation approach of Paradkar *et al.* [117] is validated using a real industrial application with 4 services and 84 operations. The authors achieved 100% requirement coverage during their experiments with a stable version of the application. During experiments, they revealed four functional bugs out of seven bugs known to be present. The authors also compared the effort and efficiency of the approach, and the results showed that the approach reduced the cost around 50% compared with manual testing used by the testing team of the company.

Endo *et al.* [105] experimented on three different examples: great common divisor process, loan approval and Nice Journey. The authors presented results of required test cases to cover all executable elements in order to prove the applicability of the used criteria. The required number of test cases for different criteria varies between 1 and 10 test cases.

Li *et al.* [112] experimented with a real service that wraps query function of a call centre. They have generated test cases for 23 out of 128 input fields of the selected service. The authors' approach generated 514 test cases within 10 min using data type constraints and boundary information. The authors highlighted the fact that this approach reduces cost vastly compared with manual test case generation. They compared their results against manual test case generation, which took 3 days to generate 293 test cases.

Belli and Linschulte [114] experimented on a real-world ISETTA service. In the experiments, 120 test cases (4 positive and 116 negative) executed and 1 of positive and 65 of negative test cases revealed a fault. Six of the revealed faults were found to be severe.

Endo *et al.* [115] experimented on an example banking system. The proposed approach achieved 100% coverage using six positive test cases in all requirements (all nodes, all edges, all uses and all Pot uses). Nineteen negative test cases are needed in order to achieve 100% coverage on the faulty pairs of ESG. Test case minimization capability of the proposed approach is also evaluated, and the approach achieved between 74% and 64% reduction in test suit size.

Yan *et al.* [106] experimented on three loan approval examples from ActiveBPEL engine. The authors performed mutation testing by introducing a bug pattern into the process and observing if any of the generated test paths covered the pattern. During the experiments, all their test path generation methods killed the mutant.

Hou *et al.* [116] performed mutation analysis on six different BPEL programs: two versions of loan approval, automated teller machine (ATM), market place, gymlocker, BPEL (1–5) in order to prove the effectiveness of their approach. In these experiments, they achieved over 98% effectiveness in test case generation and average 90% mutation score for all six programs.

Chakrabarti and Kumar [26] experimented on a real RESTful service. The authors performed daily testing routine, which took 5 min to execute the 300 test cases. Out of 300 cases, 5–10 test cases failed daily. The authors also automatically generated a test suite with 42,767 test cases, covering all possible input parameters. There were 38,016 test cases that initially failed, and in the second attempt, after bug fixes, 1781 test cases still failed.

DOI: 10.1002/stvr

The search-based test data generation approach of Blanco *et al.* [126] was ran on two example BPEL programs: loan approval and shipping service. The approach of Blanco *et al.* achieved 100% coverage for both examples, whereas random generation achieved only 99% and 75%, respectively. According to test results, the number of required test cases for coverage is much lower than for random. For loan approval, this approach achieved 100% coverage using 95% fewer test cases than random and, similarly, for shipping service, 96% less test cases.

Cavalli *et al.* [179] evaluated MBT approaches using TESFM and its variations using TRS as a case study. In the evaluation, test cases are generated using the tools (TestGen-IF, WSOTF and Web Services Atomic Transaction (WS-AT)) that are introduced in three different TESFM-based approaches. The test results provided evidence for the correctness of TRS. The authors also performed mutation testing with a single mutant, and the test cases generated by TestGen-IF were able to kill the mutant.

Li *et al.* [172] experimented on two web services: the Parlay X Conference service and the computer-supported telecommunications application routing service. The authors performed their experiments on a machine with a 3-GHz processor and 2 GB of memory. The flow analysis of Parlay X service with NINE operations took 3.00 s, and the computer-supported telecommunications application service with 10 operations took 10.00 s.

García-Fanjul *et al.* [146] experimented within a loan approval example BPEL program. The authors claimed that the approach finds a minimum number of test cases required achieve transition coverage for a given specification. The authors also mention the performance of the tool; it completes the verification in less than a second using a system with 3.0-GHz Pentium 4 processor and 2 GB of memory. The model that is used in the experiments has 32 internal states, and it was represented by a 96-byte state vector.

8.4. Discussion

Model-based testing aims to test a system using a model that sufficiently describes the expected behaviour of the SUT for testing purposes. For service compositions, a model that can sufficiently represent the process can be created, thereby allowing MBT and 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 published in the formal verification of BPEL compositions.

One of the most important advantages of formal verification for the integrators is that it can be performed offline. This is an important aspect that needs to be highlighted because it can greatly reduce the cost. Formal verification approaches can reveal errors such as unreachable parts or cause deadlocks without the need of execution. Formal verification can reveal errors that are hard to detect using testing. Thus, it is necessary for the integrator to perform both the formal verification and testing.

One important contribution of formal verification methods [142, 146, 147, 149] in testing is their ability to generate test cases. These approaches, combined with simulated testing of BPEL compositions discussed in Section 6 allow the integrator to perform automated testing with low cost.

Another important aspect that needs to be highlighted in this section is the number of tools used. In order to provide the necessary automation, both in model generation and verification, various authors have presented many tools. The tools used for verification are well-known tools such as SPIN, BLAST, NuSMV and JPF.

In the area of translation from composition languages such as BPEL to different models, many translation tools are introduced. Tools such as BPEL2IF, BPEL2oWFN and BPEL2PNML allow automation, thereby reducing the cost. There are also many other proposed methods and models that contribute towards automated BPEL translation. These publications are not mentioned in this survey to retain a focus on testing and verification.

Model-based test case generation is a well-known and exercised technique. Because most of the approaches in this section are targeting service compositions, these approaches are primarily aimed at the integrator. According to the results from the experiments, most of the proposed approaches can achieve high coverage with minimal effort. For instance, the results from approaches such as those of Endo *et al.* [105] and Blanco *et al.* [126], where required number of test cases for coverage

DOI: 10.1002/stvr

are much fewer than random, prove that they reduce the cost of testing not merely by automating the input generation process but also by reducing the number of required test cases to run on the service(s) under test.

Some of the model-based test case generation approaches [105–109, 116, 118, 126] described previously have a single common feature. In traditional MBT, models are created using requirements or specifications separately from the executable code. However, almost all of the model-based approaches described previously generate models from the executable code itself. This abstraction/translation process leads to a model that reflects the behaviour of the executable code rather than a model that reflects the expected behaviour of the system. Thus, testing using such a model will lead to testing of the translation process rather than testing of the actual system. By definition, the errors revealed using these approaches can only be those errors introduced by their translation process.

This problem does not affect the approaches using formal verification methods. In formal verification, logical properties that the SUT is checked against are not derived from the executable code. Other approaches in this section such as that of Felderer *et al.* [175] require the test model to be generated separately.

9. INTEROPERABILITY TESTING OF SERVICE-CENTRIC SYSTEMS

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 industry and the WS-I, an open industry organization, formed by the leading IT companies. The organization defined a WS-I Basic Profile [180] 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.* [181] described the possible interoperability issues regarding core Web service specifications such as SOAP, WSDL and UDDI and explained how the WS-I Basic Profile provides solutions for the interoperability issues with Web service specifications.

There are also interoperability problems that might be caused by web service toolkits such as Java Axis and .Net. For example, using dynamic data structures in services that use a certain toolkit might cause interoperability problems (because of message consumption errors) for the clients using other toolkits [182]. Interoperability problems do not occur only among different toolkits but might also occur in different versions of the same toolkit. This is also another important interoperability aspect that needs to be tested by both the developer and the certifier.

9.1. Perspectives in interoperability testing

Because the aim of the interoperability is to observe whether the services exchange messages, as expected, it can be performed by all the stakeholders in SOA. Interoperability testing of services (service protocols and interfaces) is very important for the provider and the certifier. Testing for interoperability must be included in reliability measurement. Even though most of the approaches in this section target services, there are approaches such as those of Narita *et al.* [183] and Yu *et al.* [184] that target service compositions. These approaches can only be performed by the integrator.

9.2. Interoperability testing approaches

The need for testing the interoperability among services is also recognized by researchers. For example, Bertolino and Polini [185] proposed a framework that tests the service interoperability by using service's WSDL file along with a protocol state machine diagram [119] provided by the service provider. The protocol state machine diagram carries information on the order of the operation invocations for a service. The proposed framework checks the order of the service invocations among different services and tries to point out possible interoperability problems.

DOI: 10.1002/stvr

Yu *et al.* [184] proposed an ontology-based interoperability testing approach using the communication data among services. The proposed approach captures communication data and stores it in an ontology library. The data in the library are analysed, and reasoning rules for error analysis and communication data are generated in order to run with the Jess reasoning framework [186]. Using the rules from the 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 services.

Smythe [187] discussed the benefits of the model-driven approach in the SOA context and proposed an approach that can verify interoperability of services using UML models. The author pointed out the need for UML profile for SOA that contains the interoperability specification in order to use with the proposed approach. With the use of the proposed UML profile, test cases can be generated for testing the conformance of the service's interoperability.

Similarly, Bertolino *et al.* [185] proposed a model-based approach for testing interoperability. In the proposed environment, services are tested before registration. In this approach, a service provider is expected to provide information on how to invoke a service using a UML 2.0 behaviour diagram.

Ramsokul and Sowmya [188] proposed the use of ASEHA framework to verify the service protocol's implementation against its specification. They claimed that the ASEHA framework is capable of modelling complex protocols such as Web Services Atomic Transaction (WS-AtomicTransaction) [189] and Web Services Business Activity (WS-BusinessActivity) [190]. The proposed ASEHA framework captures the SOAP messages from services, maps them into ASEHA automata and verifies the protocol implementation against its specification.

Guermouche and Godart [155] proposed a model checking approach for verifying service interoperability. The approach uses UPPAAL model checker and includes timed properties in order to check for timed conflicts among services. The approach is capable of handling asynchronous timed communications.

Betin-Can and Bultan [152] proposed the use of a model, based on HSMs for specifying the behavioural interfaces of participating services in a composite service. Betin-Can and Bultan suggested that verification of the web services that are developed using the peer controller pattern is easier to automate and proposed the use of HSMs as the interface identifiers for web services in order to achieve interoperability. Betin-Can and Bultan proposed a modular verification approach using JPF 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.* [183] proposed a framework for interoperability testing to verify Web service protocols, especially aimed at reliable messaging protocols. Narita *et al.* claimed that none of the existing testing tools aims to perform interoperability testing for communication protocols. They also highlighted the need for a testing approach that covers the reliable messaging protocols, capable of executing erroneous test cases for these protocols. As a result, their framework is capable of creating test cases containing erroneous messages by intercepting messaging across services.

Passive testing is the process of monitoring the behaviour of the SUT without predefining the input(s) [191]. The first passive testing approach for Web services was proposed by Benharref *et al.* [192]. This EFSM-based approach introduces an online observer that is capable of analysing the traces and reporting faults. The observer also performs forward and backward walks on the EFSM of the SUT in order to speed up the state recognition and variable assignment procedures.

Passive conformance testing of EFSMs was proposed by Cavalli *et al.* [191] where testing artefacts called 'invariants' enable testing for conformance. These invariants contain information on the expected behaviour of the SUT used in testing traces. Several authors extended this research to Web services domain.

For example, Andrés *et al.* [193] proposed the use of passive testing for service compositions. The proposed invariants contain information on the expected behaviour of services in the composition and their interaction properties. The proposed passive testing approach checks local logs against invariants in order to check for the absence of prescribed faults. Morales *et al.* [194] proposed a set

DOI: 10.1002/stvr

of formal invariants for passive testing. In this approach, time-extended invariants are checked on collected traces. The approach uses a tool called TIPS that enables passive testing.

Cao *et al.* [195] also proposed a passive testing approach for service compositions. The proposed approach enables both online and offline verification using constraints on data and events called security rules. The security rules are defined in the nomad language. The authors also presented a tool that automates the passive testing for behavioural conformance called Runtime Verification engine for Web Service (RV4WS).

9.3. Experimental results

Narita *et al.* [183] performed experiments on Reliable Messaging for Grid Services version 1.1, an open source implementation of WS-Reliability 1.1. The framework performs coverage-based testing in order to check for conformance to its specifications. It also performs application-driven testing to check for interoperability. The errors introduced by the framework include losing a package, changing massage order and sending duplicate messages. During coverage testing, the framework tested 180 out of 212 WS-Reliability items and was unable to find any errors but raised three warnings. During application-driven testing, four errors were revealed and four warnings were raised.

Betin-Can and Bultan [152] experimented on the travel agency and the order handling examples. For the travel agency, different peers of the program took between 4.61 and 9.72 s for interface verification. The resources used in this experiment ranged between 3.95 and 19.69 MB of memory. Synchronizability analysis of the travel agency (which was 8911 states) took 0.38 s and used 5.15 MB of memory. Order handling interface verification for peers took between 4.63 and 5.00 s and used 3.73–7.69 MB of memory. Synchronizability analysis of order handling (which was 1562 states) took 0.08 s and used 2.01 MB of memory.

9.4. Discussion

As stated, interoperability is one of the major potentials of Web services. Web services must be tested for interoperability in order to achieve this potential. Interoperability issues that are caused by different versions of the protocols such as SOAP are addressed by industry in WS-I. However, other challenges require approaches such as those of Tsai *et al.* [196] and Bertolino *et al.* [185] 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.

The approaches in this section are divided into three main groups. The first group aims to verify service protocols, such as the work of Narita *et al.* [183] and Ramsokul and Sowmya [188]. The second group verify interfaces and communication among services, such as the work of Betin-Can and Bultan [152], Smythe [187] and Yu *et al.* [184]. The third group are the passive testing approaches, such as the work of Andrés *et al.* [193], Morales *et al.* [194] and Cao *et al.* [195].

The approaches in this section can also be grouped in terms of their cost. The approaches that use formal verification and passive testing approaches will reduce the cost of testing for the integrator. Formal verification approaches cover service and protocol verification respectively. Using them together allows a complete offline interoperability testing for the integrator. The passive testing approaches will provide the integrator to ability to detect real-world usage faults. For the approaches where test cases are generated, the cost can be higher. The passive testing approaches increase the cost of testing for the integrator because of the effort required to create the necessary invariants.

10. INTEGRATION TESTING OF SERVICE-CENTRIC SYSTEMS

Integration testing is crucial in most fields of engineering to make sure all the components of a system work together as expected. The importance of performing integration testing is also well established in software engineering. Because the idea behind SOA is to have multiple loosely coupled and interoperable distributed services to form a software system, integration testing in SOA is at least as important. With the integration testing performed, all the elements of a ScS can be tested including services, messages, interfaces and the overall composition.

DOI: 10.1002/stvr

Bendetto [197] defined the difference between integration testing of traditional systems and ScS. Canfora and Di Penta [7] pointed out the challenges in integration testing in SOA. According to Bendetto and Canfora and Di Penta, the challenges of integration testing in ScS are the following:

1. Integration testing must include the testing of services at the 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 because of the number of required service calls.

10.1. Perspectives in integration testing

As might be expected, integration testing is only performed by the integrator. The rest of the stakeholders are not capable of performing integration-oriented approaches because of the lack of observability.

Most of the approaches in this section target service compositions using static binding. By contrast, for dynamic SOA, performing integration testing can be very challenging because of ScS's configuration being available only at run-time (this problem is referred as the 'run-time configuration issue' in the rest of this paper). In dynamic SOA, the integrator needs to test for all possible bindings, which can increase the cost of testing greatly.

10.2. Integration testing approaches

One of the earliest works on integration testing of Web services is the Coyote framework of Tsai *et al.* [198]. Coyote is an XML-based object-oriented testing framework that can perform integration testing. Coyote is formed of two main components: a test master and a test engine. The test master is capable of mapping WSDL specifications into test scenarios, generating test cases for these scenarios and performing dependency analysis, completeness and consistency checking. The test engine, on the other hand, performs the tests and logs the results for these tests.

In software development, there is a concept called continuous integration (CI) [199]. 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.* [200] proposed a simulation framework that addresses the service availability problem by 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.* claimed that the proposed surrogates are more flexible than the common simulation methods such as stubs and mocks and that the simulation is platform independent.

Liu *et al.* [201] also proposed a CIT approach with which executable test cases carry information on their behaviour and configuration. In the proposed approach, integration test cases are generated from sequence diagrams. The authors also introduced a test execution engine to support this approach.

Peyton *et al.* [202] proposed a testing framework that can perform 'grey-box' integration testing of composite applications and their underlying services. The proposed framework is implemented in Testing and Test Control Notation version 3 [203], a European Telecommunications Standards Institute standard test specification and implementation language. It is capable of testing the composite application behaviour and interaction among participating services. The framework increases the visibility and the control in testing by inserting test agents into a service composition. These agents are used in analysing HTTP and SOAP messages between the participating services.

Mei *et al.* [204] addressed 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.* created a model that combines these two graphs called X-WSBPEL. Data flow testing criteria based on def–use associations in XRG

DOI: 10.1002/stvr

were defined by Mei *et al.*, and with the use of these criteria, data flow testing can be performed on the X-WSBPEL model.

De Angelis *et al.* [205] proposed a model checking integration testing approach. Test cases are derived from both orchestration definition and specification of the expected behaviour for the candidate services. The authors also presented a tool that supports this approach.

Tarhini *et al.* [128] addressed the issue of service availability and the cost of testing. Tarhini *et al.* solved these problems by finding suitable services before the integration process and using only the previously selected services according to their availability. In this approach, testing to find suitable services is accomplished in four stages. The first stage is the 'find stage' in which candidate services from a service broker are found. In the second stage, selected services are tested for their correct functionality. At the third stage, each service is tested for interaction as a stand-alone component, and if it passes this stage, it is tested for interactions with the rest of the components. When a 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 for the automation of testing.

Yu *et al.* [206] addressed the interaction problems within OWL-S compositions. The approach of Yu *et al.* tests interaction among participating web services using interaction requirements. Yu *et al.* proposed an extension to existing OWL-S models to carry these requirements.

There are also previously mentioned approaches that are capable of performing integration testing. For example, the ASTRAR framework of Tsai *et al.* [66], and the proposed Enhanced UDDI server [196] are also capable of performing integration testing. Similarly, the model-driven testing approach of Lenz *et al.* [58] can be used for integration testing.

10.3. Experimental results

Huang *et al.* [200] experimented on an HR system that is transformed into a service. In this system, there are 17 components in the business layer with 58 interfaces and 22 components in the data layer with 22 interfaces. During the pure simulation without real components, seven bugs were identified as caused by issues such as reference to a wrong service, interface mismatch and missing service. During real component tests (includes surrogates as well), three bugs were identified for five components.

Mei *et al.* [204] experimented on the eight popular BPEL examples. The authors created mutants by injecting faults into three different layers in the composition BPEL, WSDL and XPath. Test sets created by the approach achieved almost 100% coverage in all test criteria considered. The authors also compared their fault detection rates with random testing. Overall, the minimum detection rates for this approach are between 53% and 67%, whereas random only achieved 18%. Mean rates of fault detection rates for this approach are between 92% and 98%, whereas random achieved 73%. The authors also investigated the performance of the approach. It took between 0.45 and 1.2 s for generating test sets with a 2.4-GHz processor and 512-MB memory.

Liu *et al.* [201] experimented on two synthetic examples: an HR system and a meeting room management system. The approach revealed 22 bugs in the HR system and seven in the meeting room system. The approach revealed faults in categories such as incorrect method calls, incorrect parameter passing, configuration problems and interface/function mismatches.

10.4. Discussion

Integration testing is one of the most important testing methodologies for SOA. The challenges that the integrator faces during integration testing are addressed by some of the approaches mentioned in this section such as the frameworks of Tarhini *et al.* [128], Huang *et al.* [200] and Liu *et al.* [201].

The CI-based integration testing of Huang *et al.* [200] can be very useful by starting testing early. The ability to use surrogate services can also help in reducing the cost of testing. Because surrogate services can be generated automatically, using them does not increase the overall cost. The only handicap of this approach might be finding/generating suitable Web service specifications to be used in surrogate generation. One other issue that can increase the cost of testing is the lack of automated test case generation within the framework.

DOI: 10.1002/stvr

Liu *et al.* [201] partly automated test case generation in CIT using sequence diagrams. This approach makes use of the work of Huang *et al.* and is able simulate unavailable components. As a result, it has the same restrictions as the work of Huang *et al.* regarding the service simulation. The approach's ability to verify execution traces using object comparison and expression verification is the main advantage of this approach.

The approach of Mei *et al.* [204] addresses a problem that is overlooked by many developers. Integration issues that can be caused by XPath are an important problem in service compositions that need to be tested. The results from their experiments proved the effectiveness of their approach in revealing these problems.

Almost all of the approaches discussed previously will have problems adapting to dynamic environments. For example, the approach of Tarhini *et al.* [128] might be rendered inapplicable because of not being able to know the services available at run-time and not being able choose the service to bound at run-time. On the other hand, the approaches of Huang *et al.* [200], Peyton *et al.* [202], Tsai *et al.* [198] and Mei *et al.* [204] might become more expensive to perform.

11. COLLABORATIVE TESTING OF SERVICE-CENTRIC SYSTEMS

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

Challenges involving testing ScS were identified by Canfora and Di Penta [7], some of which require collaborative solutions. These challenges that might require collaborative solutions are the following:

1. users not having a realistic test set;
2. users not having an interface to test ScS; and
3. the need for a third-party testing and QoS verification rather than testing by each service user.

11.1. Perspectives in collaborative testing

Collaborative testing requires collaboration among stakeholders. The proposed approaches described in this section seek to establish a collaboration between the developer and the integrator. Some of the approaches include a third party in order to increase testability.

11.2. Collaborative testing approaches

Tsai *et al.* [207] proposed a cooperative validation and verification model that addresses these challenges instead of the traditional independent validation and verification. One example of this collaborative testing approach is the proposed enhanced UDDI server of Tsai *et al.* [196]. This UDDI server further enhances the verification enhancements in UDDI version 3 [15]. These proposed enhancements include the following:

1. The UDDI server stores test scripts for the registered services.
2. The UDDI server arranges test scripts in a hierarchical tree of domains and subdomains.
3. The UDDI server has an enhanced registration mechanism called check-in. The check-in mechanism registers a service if it passes all test scripts for its related domain and subdomain.
4. The UDDI server has a new mechanism before the client is able to use the selected service called *check-out*. This mechanism allows the client to test any service before using it with the test scripts from service's associated domain.
5. The UDDI server includes a testing infrastructure that allows remote WST.

In the proposed framework, the provider, as suggested by Canfora and Di Penta, can also provide test scripts to point out qualities of the 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.

DOI: 10.1002/stvr

Bai *et al.* [208] also proposed a contract-based collaborative testing approach that extends the enhanced UDDI proposal of Tsai *et al.* Bai *et al.* proposed 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.* suggested two types of contracts for the DCV&V approach: test collaboration contracts, which enforce the collaboration among the test brokers, and testing service contract, which is used for contract-based test case generation.

The proposed test broker of Bai *et al.* provides a test case repository for test cases and test scripts, collects test results and maintains defect reports and service evaluations. The test broker can generate and execute test cases as well. Bai *et al.* suggested that, with the use of the DCV&V architecture, multiple test brokers can become involved in the testing process. The decentralized architecture allows flexible and scalable collaborations among the participants.

Zhu [209, 210] proposed another collaborative approach. In Zhu's approach, service developers or third stakeholder testers provide testing services that help with testing. Zhu also proposed a testing ontology that is based on a taxonomy of testing concepts called the STOWS. The proposed ontology aims to solve the issues related to the automation of test services.

Bartolini *et al.* [211] introduced a collaborative testing approach that 'whitens' the ScST by introducing a new stakeholder called TCov that provides the tester with coverage information. The approach of Bertolini *et al.* requires the service provider to insert instrumentation code inside the service in order to provide TCov with coverage information. This information is then analysed by TCov provider and is made available to the tester as a service.

Eler *et al.* [212] proposed an approach to improve web service testability. The proposed approach provides the tester with an instrumented version of the SUT as a testing service. The testing service is instrumented by the developer, aiming to provide the tester with coverage information and other testing-related metadata. The authors also presented a web service that automatically generates the testing service from Java byte code called JaBUTiWS and also another tool WSMTS, which supports the testing process.

11.3. Experimental results and discussion

Collaborative testing approaches aim to solve some of the challenges involved in ScST, for example, having an adequate test suite by allowing service providers to provide test suits or having a third stakeholder tester providing testing interfaces for the service consumers. The claimed benefits of these approaches justify collaborative testing of ScS.

The proposed testing approach of Tsai *et al.* [196] provides many benefits, such as increasing the quality of testing by providing realistic test cases from the provider. This approach also reduces the number of test runs by testing services before the binding process has been completed. The approach also reduces the cost of testing through automation. 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 services.

The contracts of Bai *et al.* [208] allow the test provider to supply 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 stakeholders. Generally, contracts aim to enforce the correct functionality of a system. The contracts of Bai *et al.* additionally enforce collaboration among stakeholders. However, although contract-based collaboration enforcement may be effective, the cost of generating contracts might be discouraging.

The main advantage of Zhu's [209, 210] 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 errors that might happen during testing process or a decrease in the service's performance. The proposed environment also reduces 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

DOI: 10.1002/stvr

stakeholder testing services, which require no testing. Using third stakeholder testing services might increase the cost vastly because of the costs of testing services.

Bartolini *et al.* [211] provided experimental results from application of their TCov environment. The authors highlighted an important aspect of using TCov that it enables the tester to receive information on test coverage without violating the SOA principles. The authors also suggested that knowing coverage results from different test cases can greatly help the tester to perform better tests. The main advantage of this approach is that it can be easily used in the current Web services environment. The main disadvantage of the TCov environment is the introduction of a third party into the testing process, which increases the cost of testing.

Eler *et al.* [212] provided performance analysis results based on the overhead created by the proposed approach. The authors measured the execution time of a test suite on the original service and compared the execution time of the same test suite with the testing service. The overhead added by the approach increased execution times 2.65% outside the testing session and 5.26% in the testing session. This approach provides the benefits of the approaches of Zhu [209, 210] and Bartolini *et al.* [211] without the need for a certifier. The main disadvantage of this approach, at present, is that it can only be applied to Java web services.

12. TESTING SERVICE-CENTRIC SYSTEMS FOR QOS VIOLATIONS

QoS and its requirements have been discussed for many years in areas such as networking. A general QoS definition is given by Campanella *et al.* [213] as

> QoS (Quality of Service) is a generic term which takes into account several techniques and strategies that could assure application and users a predictable service from the network and other components involved, such as operating systems.

QoS for is Web services defined as

> Quality of Service is an obligation accepted and advertised by a provider entity to service consumers.

in W3C Web Services Glossary [214].

In the literature, QoS generally refers to nonfunctional properties of Web services such as reliability, availability and security. A broad definition of QoS requirements for Web services is given by Lee *et al.* [215]. This definition includes performance, reliability, scalability, capacity, robustness, exception handling, accuracy, integrity, accessibility, availability, interoperability and security.

In SOA, the need for QoS is highlighted by the two main questions:

1. the selection of the right service; and
2. the provision of guarantees to the service consumer about service performance.

As mentioned, one of the most important benefits of SOA is the ability to use services from other businesses. As a consequence, the consumer often has the problem of choosing a suitable service. In such an environment, businesses must have the ability to distinguish their services from the competition. On the other hand, the consumers must have the ability to compare and choose the best service for their needs. As a result, QoS ratings must be published by the provider and be accessible to the consumer within the SOA environment.

The answer to the second question is a well-known concept called the service-level agreement (SLA). SLA is an agreement/contract between the provider and the consumer(s) of a service that defines the expected performance of a service at defined levels. SLAs might also include penalty agreements for service transactions or usage periods where the service performs below an agreed level.

It is hard to define a standard SLA that fits all kinds of available services. Services covered by an SLA also need to be tested for their conformance to the SLA. The service subsequently needs to be monitored during its normal operations to check SLA conformance.

The importance of QoS in SOA and the problems surrounding it has led to the service standard called WS-Agreement [216]. WS-Agreement is a specification language aimed at standardizing the overall agreement structure. The language specifies domain-independent elements that can be extended to any specific concept.

DOI: 10.1002/stvr

Because of its significance in SOA, QoS testing is as important as functional testing. The difference in QoS testing is that it needs to be performed periodically and/or the service needs to be monitored for SLA conformance.

12.1. Perspectives in QoS testing

QoS testing of Web services is performed by the provider (or the certifier if requested). The integrator can perform QoS testing on compositions to reveal possible SLA violations. QoS testing at the integrator side can be considered as a static worst-case execution time analysis using the nonfunctional parameters of services in the composition. Stub services can also be generated with given QoS parameters in order to perform simulated testing. The run-time configuration issue does not affect the integrator neither in simulated QoS testing nor in static analysis. For the dynamic SOA, the integrator can use service selection constraints rather than services' actual QoS parameters.

The cost of QoS testing can be higher than traditional testing. The two primary cost drivers are the cost of service invocation and the need to generate real-usage test data. The primary issue is due to the requirement of running each test case several times during testing period. The reason for multiple test case executions is that the average of the results from multiple test runs provide a more realistic QoS score than a single test. QoS scores also need to be updated periodically from the monitoring data or results from tests.

12.2. QoS testing approaches

The oldest research publication on QoS testing of services is by Chandrasekaran *et al.* [217]. The authors proposed an approach that is used to test performance of services and simulation-based evaluation for service compositions. The authors introduced their tool called Service Composition and Execution Tool, that allows service composition as well as evaluation. Simulation in the tool is handled by JSIM, a Java-based simulation environment.

Di Penta *et al.* [218] proposed testing for SLA violations using search-based methods. In this approach, inputs and bindings for ScS are generated using GAs. The generated inputs and bindings aim to cause SLA violations. For service compositions, a population to cover the expensive paths are evolved by the authors' GA that try to find violating conditions. The approach also monitors QoS properties such as response time, throughput and reliability. Monitored parameters are used as fitness function in test case selection.

The regression testing approach of Di Penta *et al.* [219] aims to test nonfunctional properties of services. It allows the developer to publish test cases along with services that are used in the initial testing. In this approach, assertions for QoS are used for testing SLA conformance. These assertions are generated automatically with the executions of test cases.

Gu and Ge [220] proposed another search-based SLA testing approach. Similar to the approach of Di Penta *et al.*, a CFG is derived from service composition and a QoS analysis is performed on each path. Test cases are generated around the maximum and minimum SLA constraints. The difference in this approach is the added users' experience, which is included in defining QoS sensitivity levels.

Palacios *et al.* [221] proposed a partition testing-based SLA testing approach. In the proposed approach, test specifications are generated from information in SLAs that are specified in the form of WS-Agreement specifications. The category-partition method is applied to these specifications to generate test cases that aim to reveal SLA violations.

The need for test beds in SOA is mentioned previously for functional testing. Bertolino *et al.* [222] suggested that test beds must also be able to evaluate QoS properties in SOA. The authors proposed a test bed that allows functional and nonfunctional testing of service compositions. The proposed test bed can perform testing without the need of invoking outside services using service stubs. Stubs are generated using a tool called Puppet (Pick Up Performance Evaluation Test-bed) [223]. The Puppet generates service stubs using QoS parameters in SLAs expressed in WS-Agreement specification. In this approach, functional models of the stubbed services are expected to be provided by the provider in the form of STS. The information from STS and SLA allow generating stubs with both functional and nonfunctional properties.

DOI: 10.1002/stvr

Another example to test beds that can perform offline testing is the MaramaMTE [225] tool of Grundy *et al.* [224]. This tool provides similar functionality to the test bed of Bertolino *et al.*. The most important functionality of this test bed is it allows testers to perform offline performance tests on service compositions. In this approach, service stubs are created from composition models such as Business Process Modeling Notation and ViTaBal-WS. Unfortunately this approach does not use SLAs to inform the tester about the possible violations.

Driss *et al.* [226] proposed a QoS evaluation approach for service compositions using discrete-event simulation. Discrete-event simulation is performed with a model that represents the operation of a system as a chronological sequence of events. The authors proposed a model that includes BPEL activities and network infrastructure. Simulations in this approach are performed using the NS-2 simulator.

Pretre *et al.* [227] proposed a QoS assessment framework using MBT called iTac-QoS. In this approach, the provider is expected to provide a UML-based test model formed of three diagrams. These diagrams contain a service interface, temporal evolution and input data. Functional test data are generated from the model. The framework uses a categorization of tests, goals and results. In order to automate categorization process, requirements in the form of OCL are provided. Total automation is one of the highlights of this approach.

Yeom *et al.* [228] introduced a QoS model and a testing mechanism to test service manageability quality for this model. In this approach, a set of manageable interfaces are provided along with services. These interfaces provide functions to get internal information about services to increase observability or to provide change notifications.

12.3. Experimental results

Di Penta *et al.* [218] experimented on a synthetic audio processing workflow with four services and an existing image manipulation service. The authors compared their GA approach against random search to prove its effectiveness. In their experiments, GA significantly outperformed random search. Both their black-box and white-box approaches proved to generate inputs and bindings that can violate SLAs. Their comparisons between these two approaches showed that the white-box approach takes less time than black-box approach to find a solution that violates QoS constraints.

Di Penta *et al.* [219] experimented on a synthetically generated services based on five releases of dnsjava. The most important finding in their experiments was the overall QoS increase with new releases. The only version that had lower QoS results among the new versions was the version with new features. The authors also found out that SLAs might be violated by the new versions if they cover the newer versions.

Gu and Ge [220] performed their experiments on a synthetic service composition. Their experiments showed that their approach is able to identify QoS-risky paths and generate test cases that can violate QoS constraints. They have also compared their GA approach to a random search similar to Di Penta *et al.*, and according to their results GA outperformed the random search.

Driss *et al.* [226] experimented on a travel planner composite service example. The authors focussed solely on the response time parameter. The results from their simulation was able to identify the fastest service method and also methods that have problems with response times.

12.4. Discussion

One of the main topics in SOA is QoS and SLAs. In order to establish a healthy SOA environment, it is very important to establish a standard way of representing QoS parameters and measuring them. In SOA, it is imperative that up-to-date QoS parameters are checked for their conformance to SLAs. This necessity highlights the importance of QoS testing and SLA conformance. The need for updating QoS parameters and SLAs with each new service version also increases the need for QoS testing.

One of the main problems the integrator faces in QoS testing is the need for testing all possible bindings. This problem is caused by the dynamic nature of SOA when the integrator does not know which services will be selected and invoked. Even though expected QoS parameters in SLAs give an idea about services' performance and enable static/simulated QoS analysis, these parameters do not

DOI: 10.1002/stvr

reflect the network performance. For example, determining network latency for all possible users can be unrealistic.

Existing work in QoS testing can be classified into two main areas according to the way they are performed. These two main categories are simulation and real testing. The difference between these two methods are the use of service stubs rather than invoked services.

The main advantage of using stubs is its low cost. Because QoS testing can be very expensive, using stubs is an ideal way to reduce cost. Reducing cost is invaluable for the integrators and allows them to run more tests to reveal additional possible SLA violations. Another benefit of simulation is its ability to adapt dynamic SOA. In simulation, QoS ratings of existing services can be used to generate a probabilistic SLA for the whole composition.

Although simulation can be a solution to adaptation issues in QoS, it does suffer from network performance representation. The approach of Driss *et al.* [226] addresses this issue, by including network models in their simulations. Unfortunately, this approach cannot be used in dynamic SOA environment.

Testing with real services has the advantage of getting realistic QoS ratings and finding run-time SLA violations. The weaknesses of testing with real services is the need to test for all possible bindings and the high costs involved. The similar approaches of Di Penta *et al.* [218] and Gu and Ge [220] can help reduce the cost of testing for SLA violations by detecting high-risk paths and focussing testing on these parts of service compositions.

13. REGRESSION TESTING OF SERVICE-CENTRIC SYSTEMS

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 carried out in a white-box manner [229]. 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 [229]. However, approaches such as the graph walk approach (GWA) by Rothermel and Harrold [230] that does not require access to the source code can be used in ScS regression testing (ScSRT). This survey distinguishes the regression test selection (RTS) methods that require access to source code in order to identify those RTS methods applicable to testing Web services. Because the integrator does not have source code access at the service level, RTS methods that require access to the source code are inapplicable in ScST.

According to Canfora and Di Penta [7], one of the main issues in ScSRT at the consumer side is not knowing when to perform regression testing. Because the consumer has no control over the evolution of the service, he or she might not be aware of the changes to the service. There are two possible scenarios for informing the consumer about such modifications. These scenarios are based on the provider's knowledge about the consumers.

The first scenario arises when the SUT is registered to a UDDI broker or is not a free service. The subscription service in UDDI version 3 allows automated notification of the consumers when changes are made to a service. For paid services, it is assumed that the provider has the details of the consumers through billing or service agreements. In this scenario, informing the consumers about the changes that are made to a service will not be a problem. Even so, there is still a small room for error. If the consumers are not properly informed about which methods of the service are modified, they might either perform unnecessary tests or fail to perform necessary tests.

The second scenario arises when the web service that requires regression testing is a public service with no UDDI registration and the provider does not have information about its consumers. This scenario is the most problematic one, because the consumer 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 the consumers discovering the changes, the confidence in the service might decrease because of errors or decreases in QoS.

DOI: 10.1002/stvr

Another challenge in ScSRT is the concurrency issues that might arise during testing because of the tester not having control over all participating web services. Ruth and Tu [231] discussed these issues and identified possible scenarios. They identified three different scenarios, all of which 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.

13.1. Perspectives in regression testing

Regression testing is another essential testing methodology that can be performed by all the stakeholders in SOA. ScSRT for the developer is the same as traditional regression testing.

The ScSRT approaches in this section are divided into two categories: regression testing for single services and the service compositions. The approaches for service compositions such as those of Ruth and Tu [232], Liu *et al.* [233], Mei *et al.* [234] and Tarhini *et al.* [127] are aimed to be performed by the integrator. Approaches such as those of Tsai *et al.* [235], Di Penta *et al.* [219] and Hou *et al.* [236] are expected to be performed by the integrator and the certifier. The approach of Lin *et al.* [237] can only be performed by the developer.

13.2. Regression testing approaches

As explained earlier, the RTS method of Rothermel and Harrold is used by many ScSRT researchers. The proposed approaches by the researchers usually differ in the method of the CFG generation.

Ruth and Tu [232] proposed 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 proposed 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 [238]. The required CFG needs to be constructed at the 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.

Ruth *et al.* [239] also proposed an automated extension to their RTS technique that tackles the concurrency issues that might arise during ScSRT. This approach helps in solving the multiple modified service problem by using *call graphs* [232]. 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 took the situation into consideration where a service makes multiple calls to different services in parallel.

Lin *et al.* [237] proposed another GWA-based regression testing approach where CFGs are created from Java interclass graph [240]. A framework that performs RTS on the transformed code of a Java-based web service was also proposed by Lin *et al.* The code transformation can be performed only in Apache Axis framework [241]. The proposed approach uses the built-in WSDL2Java-generated [242] classes both on the server and the tester 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 Java interclass graph on which the GWA can be performed. Compared with the previously presented approaches, the approach of Lin *et al.* is able to generate CFGs in an automated fashion without the knowledge of internal behaviour of the web service. The main limitation of this approach is its restricted application to Java-based web services.

Liu *et al.* [233] addressed the issues that occur because of concurrency in BPEL regression testing. Lui *et al.* proposed 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.

DOI: 10.1002/stvr

Tarhini *et al.* [127] proposed another model-based regression testing approach using the previously explained model in Section 8. 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; and
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 types of modifications are represented by adding or removing states or edges from the TLTS of the original system.

An approach that performs regression testing for BPEL processes was proposed by Wang *et al.* [243]. This approach uses the BFG [107] that was described in Section 8.2.1. Wang *et al.* proposed 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 that the authors claim is better suited to regression testing. Li *et al.* [244] introduced another eXtensible BFG-based ScSRT approach where test case generation and selection is based on the comparison of different versions of BPEL applications. Yang *et al.* [245] proposed an approach that aims at providing effective fault localization using recorded testing symbols. The proposed symbols contain the test step number and the service interface information. The approach is supported by test scripts that contain information test data and test behaviour.

Mei *et al.* [234] proposed 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.

Mei *et al.* [246] also proposed another coverage model that captures BPEL process, XPath and WSDL that enables test case prioritization. In this approach, test cases can be sorted by XRG branch coverage and WSDL element coverage in addition to their BPEL branch coverage.

Athira and Samuel [247] proposed a model-based test case prioritization approach for service compositions. The approach discovers most important activity paths using a UML activity diagram of the service composition under test.

Chen *et al.* [248] also proposed a model-based test case prioritization approach based on impact analysis of BPEL processes. The authors introduced a model called BPEL flow graph (BPFG) into which BPEL processes are translated for change impact analysis. Test cases are prioritized according to the proposed weighted dependence propagation model.

Zhai *et al.* [249] proposed a test case prioritization technique that incorporates service selection for location-aware service compositions. In order to overcome the potential issues caused by dynamic service selection, the authors proposed a service selection phase. In this phase, a service known to behave as expected is selected and bound to the composition before the testing process. The authors introduced a concept called point of interest-aware prioritization technique, which is more effective than traditional input-directed techniques for location-aware services.

The need for a visual testing tool was addressed by Pautasso [250]. The author proposed 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 [251]. One of the most important features of JOpera is its ability to reflect changes and allow better regression testing. JOpera separates the composition model from the service descriptions. Therefore, it can test both independently. JOpera regression testing starts with registry query to discover existing test cases. The approach uses snapshots to capture and compare execution states with expected states. Data flow information helps with fault localization.

Tsai *et al.* [235] proposed a model-based adaptive test case selection approach that can be applied to both regression testing and group testing. This approach defines a model called the coverage relationship model 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.* defined multiple rules that guarantee the selection of the most potent test cases and proved that the less potent test cases never cover the more potent test cases. Tsai *et al.* claimed that this approach can be applied to regression testing when a new version of a service with the same specifications is created.

DOI: 10.1002/stvr

In ScSRT, test case management has other test case prioritization considerations such as service access quotas. Hou *et al.* [236] addressed the issue of quota constraints for ScSRT. The quota problem might occur when performing regression testing on web services with a limited number of periodic accesses. The use of 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.* [236] proposed 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 on the basis of time slots that suit the user's web service access quotas. The proposed test case prioritization approach is based on a multiobjective selection technique that defines an objective function that aims to attain maximum coverage within the quota constraints of services.

Di Penta *et al.* [219] proposed a collaborative regression testing approach that aims to test both functional and nonfunctional properties of web services. The approach of Di Penta *et al.* 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.

13.3. Experimental results

Mei *et al.* [234] experimented on eight BPEL examples and randomly generated 1000 test cases for these programs. With the use of these test cases, 100 test suites with average 86 test cases were generated. The authors claimed that their black-box testing approaches can achieve similar fault detection rates to white-box testing approaches.

Mei *et al.* [246] also used the same case study to evaluate their coverage-based approach. The authors proposed 10 different techniques for prioritization. All 10 techniques were found to be more effective than random prioritization. The results also suggest that the techniques augmenting additional coverage information from WSDL elements and XRG provide more effective prioritization.

Di Penta *et al.* [219] experimented on five synthetically generated web services based on five releases of dnsjava. The outputs are analysed from two different perspectives. The first perspective is the comparison against all other service releases. The second perspective is checking the output from a single service. The results using the first perspective highlighted that this method can easily detect errors that arise with a new release of a web service.

Zhai *et al.* [249] experimented on a real-world service composition: City Guide. The proposed service selection method reduced service invocations by 53.18%. The point of interest-aware techniques outperformed the input-guided techniques in invocation reduction.

13.4. Discussion

The issues that relate to ScSRT for all the stakeholders in SOA are addressed by some of the works discussed previously. For example, one of the major issues in ScSRT is that of the integrator not having a realistic test suite. This issue was addressed by Ruth and Tu [232] and Di Penta *et al.* [219].

The regression testing approach of Ruth *et al.* 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. Furthermore, it might not be desirable to inform all integrators of a service at once and allow them to perform tests at the same time in order to avoid service disruptions.

This issue of possible service disruptions is a big problem at the provider side. Multiple services provided by the same provider might be affected from the regression testing of another service. As a result, the QoS of services other than the SUT might also be reduced.

DOI: 10.1002/stvr

Service-centric system regression testing exacerbates some of the cost-related problems, such as high cost due to service invocations. Unfortunately, none of the approaches in this section provides a complete solution to this problem. The approaches that reduce the number of test cases, such as that of Tsai *et al.* [235], can help minimize this problem. Nonetheless, a large regression test suite might require a high number of executions.

An additional concern related to the limitations in performing ScSRT due to quota restrictions was addressed by Hou *et al.* [236]. This approach does not reduce the cost but helps towards completing the test within a budget.

Some of the issues mentioned in this section and their solutions are based on static web service usage. The issues such as informing integrators about the changes to a service or quota limitations will be phased out with the coming transition to dynamic SOA. In dynamic SOA, the integrator's need for testing the system with new service versions and service disruptions due to regression testing by many integrators are expected to be eliminated. This is because, in dynamic SOA, testing for new service versions may be hampered by the lack of knowledge concerning run-time binding.

14. FUTURE OF SERVICE-CENTRIC SYSTEM TESTING

In order to predict the future of ScST, first, the future of SOA and services needs to be discussed. Web services are receiving increased attention with the switch towards Web 3.0. Many existing Web sites may transform into web services as a result of this transition [252, 253]. Existing sites such as Amazon, Google and Microsoft have transformed their businesses towards services.

Traditional Web services fail to exploit full potential of SOA because of difficulties in meeting Web service requirements [254]. One of the next goals of the services community is the establishment of a dynamic SOA. In order to bring this vision to fruition, it is expected that SWSs will become the new standard service practice [255]. Even though there are many promising initiatives for SWS, unfortunately, none of these initiatives has yet been accepted as a standard.

As mentioned by several authors [196, 208, 211], it is feasible to expect services to be tested by the provider or the certifier before registration. SLAs are also created by the provider after testing with integrator trust being established by the provider. A similar business model is used by application store providers. For example, the Apple iTunes [256] store acts as a provider for many developers while maintaining QoS for the products in the store.

It is also important to identify the current issues in ScST in order to draw a road map for future research. The open issues and the issues that require more research in ScST are the following:

1. lack of real-world case-studies;
2. solutions that can generate RTD;
3. solutions to reduce the cost of ScST;
4. solutions that improve the testability of ScS;
5. solutions that combine testing and verification of ScS; and
6. modelling and validation of fully decentralized ScS.

The last three issues in the list are already highlighted in Canfora and Di Penta's [6] survey. Even though these issues are discussed in their work, they are also included in the present survey and are discussed in more details with the information from the recent research that addresses them.

We believe that one of the most important current problems of ScST research is the lack of fully functioning and fully available real-world examples. ScST research needs case studies in order to measure the effectiveness and scalability of the proposed testing approaches. As it is mentioned in Section 2, 71% of the publications surveyed provide no experimental results. Furthermore, 18% of the papers, although they did provide experimental results, drew these results from synthetic services or compositions. Only 11% of the papers used real-world case studies (as depicted in Figure 4).

The RTD generation problem, mentioned in Section 5, is also a major problem in ScST. The importance of RTD in testing (especially in ScST) was discussed by Bozkurt and Harman [32]. An example to this problem is a web service that requires composite material specifications as input. In order to test this service, the tester will require very specialized data that existing automated test data generation techniques cannot effectively generate. In such a situation, the tester has two

DOI: 10.1002/stvr

options: either to get test data from the developer or to find data from other available resources. This scenario highlights the need for collaboration in ScST as well as the need for approaches that can use existing resources. Approaches that promote collaboration in testing are presented in Section 11. Unfortunately, testing that uses only the test cases that are provided by the developer might not provide the necessary level of assurance for other SOA stakeholders. Most of the surveyed test case and test data generation approaches are able to generate test data to perform boundary analysis or robustness testing but lack the ability to generate realistic data. The only two approaches aiming to generate RTD were proposed by Conroy *et al.* [33] and Bozkurt and Harman [32]. The low number of publications addressing this issue is an indicator to the need for more research.

Solutions that reduce the cost of testing are also required. Increased test frequency in SOA exacerbates the severity of this issue in ScST. This issue has two dimensions in ScST: cost of testing at composition level and at service level. The cost at both of these levels is increased by the integrator's need to test compositions with real services. The cost of invoking services during testing is the problem at service level. The cost at this level depends on the number of services in the composition and the size of the test suite. Simulated testing approaches for service compositions [49, 69, 71] can help with validation. However, they do not eliminate the need to test with real services.

The cost of testing at the service level are twofold: service disruptions due to testing and business transactions that might be required to occur during testing. Unfortunately, there is no existing mechanism to avoid these costs. Approaches such as that of Zhu *et al.* [210] may provide a solution in which testing is performed on services with the same functionality rather than the actual service itself. These approaches provide benefits similar to simulated testing although they also carry the same disadvantages.

The need for testability improvement is one of the issues that almost all authors of SCST agree upon. Although the proposed solutions to this problem look at the issues from different perspectives, they can be divided into two categories. One of the two categories is the use of contracts that provide information such as preconditions or postconditions. The second one is the use of models or new stakeholders that provide coverage information to the tester. As mentioned in Section 5.2, the effort required to create contracts or external models can be discouraging for the developer. There is also the problem of the adoption of a standard model and its integration into Web service specifications. Automated model comparison can be useful in order to provide the tester with test coverage information. Models that are built from tests can be compared with the models created by the tester. An example solution to this issue is the approaches of Bartolini *et al.* [211] and Eler *et al.* [212] with which coverage information is provided with the involvement of another stakeholder. Salva and Rabhi [257] discussed problems regarding observability and controllability of ScS.

The main aim of the verification approaches presented in this survey is checking for interface and protocol conformance. Monitoring is generally proposed to verify QoS aspects of services. However, monitoring-based approaches such as passive testing [192–194] provide run-time fault detection and a degree of fault localization using artefacts such as invariants and contracts. Unfortunately, the proposed monitoring approaches primarily check service interactions for prescribed faults.

Decentralized system testing and service choreographies are different to testing service compositions. According to Canfora and Di Penta, flexibility of service choreographies brings new challenges to testing and monitoring. Bucchiarone *et al.* [258] also stated the problem of formalizing choreographies into standard models and testing them. Recent works [259–262] address the issues of testing of service choreographies. The number of related publications in this subject compared with the number of publications in testing service compositions shows the need for more research.

15. CONCLUSION

Service-oriented computing changed the business understanding of the whole software industry. However, the change from traditional software to services and the service usage remains sluggish when compared with early expectations. One of the most important issues that inhibits the wider use of Web services is the issue of trust. One of the effective solutions to this trust issue is testing.

DOI: 10.1002/stvr

Testing Web services is more challenging than testing traditional software because of the complexity of Web service technologies and the limitations that are imposed by the SOA environment. Limited controllability and observability render most existing software testing approaches either inapplicable or ineffective. Some of the technological benefits of SOA such as late binding and dynamic service selection also increase the level of testing challenge.

The present survey focusses on testing techniques and approaches that have been proposed for testing Web services. Fundamental functional testing methodologies such as unit testing, regression testing, integration testing and interoperability testing of Web services are covered in the present survey. Other testing methods such as MBT, 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 stakeholders participate in testing are categorized under collaborative testing. The present survey also covered work on testing for QoS violations.

As Web services increasingly attract more attention from the industry and the research communities, new issues involving ScST are being identified. Some of the previously identified issues are addressed by the approaches discussed in the present survey, whereas others still await effective solutions. Several of the unaddressed issues in ScST need new and more efficient solutions, thus suggesting new opportunities and challenges. The areas in ScST that require more research are also identified in order to provide researchers a road map for future work.

ACKNOWLEDGEMENTS

Lorna Anderson and Kathy Harman assisted in proofreading. The authors sent the present survey 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.

Prof. Harman is supported, in part, by EPSRC grants: CREST Platform Grant, SLIM (SLIcing state based Models) and SEBASE (Software Engineering By Automated SEarch) and by donations from Sogeti and Motorola Inc. Dr Hassoun is supported by the EPSRC CREST Platform Grant (EP/G060525, EP/F059442, EP/D050863).

REFERENCES

1. AppLabs. Web Services Testing a Primer, May 2007. Available from: http://www.docstoc.com/docs/4248976/Web-Services-Testing-A-Primer [last accessed 27 March 2012].
2. Wintergreen Research, Inc. Services Oriented Architecture (SOA) Market Opportunities, Strategies, and Forecasts, 2006 to 2012, 2006. Available from: http://www.wintergreenresearch.com/reports/soa.html [last accessed 27 March 2012].
3. International Data Corporation (IDC). Available from: http://www.idc.com/ [last accessed 27 March 2012].
4. CBDI Forum, 2002. Available from: http://www.cbdiforum.com/ [last accessed 27 March 2012].
5. Canfora G, Di Penta M. SOA: testing and self-checking. In *Proceedings of the International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, Bertolino A, Polin A (eds): Palermo, Italy, June 2006; 3–12.
6. Canfora G, Di Penta M. Service-oriented architectures testing: a survey. In *Software Engineering*, Lucia A, Ferrucci F (eds). Springer-Verlag: Berlin, Heidelberg, 2009; 78–105.
7. Canfora G, Di Penta M. Testing services and service-centric systems: challenges and opportunities. *IT Professional* March 2006; **8**(2):10–17.
8. Tsai WT, Wei X, Chen Y, Paul R. A robust testing framework for verifying Web services by completeness and consistency analysis. In *SOSE '05: Proceedings of the IEEE International Workshop,* Beijing, China. IEEE Computer Society: Washington, DC, USA, October 2005; 151–158.
9. Bai X, Dong W, Tsai WT, Chen Y. WSDL-based automatic test case generation for Web services testing. In *SOSE 2005: Proceedings of the IEEE international Workshop on Service-Oriented System Engineering,* Beijing, China. IEEE Computer Society: Washington, DC, USA, October 2005; 207–212.
10. Tsai WT, Bai X, Chen Y, Zhou X. Web service group testing with windowing mechanisms. In *SOSE '05: Proceedings of the IEEE International Workshop,* Beijing, China. IEEE Computer Society: Washington, DC, USA, October 2005; 221–226.
11. Web Services Description Language (WSDL 1.1). Available from: http://www.w3.org/TR/wsdl [last accessed 27 March 2012].
12. OWL-S: Semantic Markup for Web Services. Available from: http://www.w3.org/Submission/OWL-S/ [last accessed 27 March 2012].
13. Web Service Modelling Ontology (WSMO). Available from: http://www.wsmo.org/ [last accessed 27 March 2012].

DOI: 10.1002/stvr

14. García-Fanjul J, de la Riva C, Tuya J. 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,* Windsor, UK. IEEE Computer Society, August 2006; 127–130.
15. UDDI Spec Technical Committee Draft. Available from: http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm [last accessed 27 March 2012].
16. SOAP Version 1.2. Available from: http://www.w3.org/TR/soap12-part1/ [last accessed 27 March 2012].
17. Brenner D, Atkinson C, Hummel O, Stoll D. 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,* Newport Beach, California, USA, Vol. 0. IEEE Computer Society: Washington, DC, USA, June 2007; 114–121.
18. Bloomberg J. Web Services Testing: Beyond SOAP, September 2002. Available from: http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci846941,00.html. [last accessed 27 March 2012].
19. Bartolini C, Bertolino A, Marchetti E, Polini A. WS-TAXI: a WSDL-based testing tool for Web services. In *ICST '09: Proceedings of the International Conference on Software Testing Verification and Validation,* Denver, Colorado, USA. IEEE Computer Society: Washington, DC, USA, 2009; 326–335.
20. Bertolino A, Gao J, Marchetti E, Polini A. Automatic test data generation for XML schema-based partition testing. In *AST '07: Proceedings of the 2nd International Workshop on Automation of Software Test*. IEEE Computer Society Washington, DC, May 2007; 4.
21. Hanna S, Munro M. Fault-based Web services testing. In *ITGN: 5th International Conference on Information Technology: New Generations (ITNG 2008),* Las Vegas, NV, USA. IEEE Computer Society, April 2008; 471–476.
22. Li ZJ, Zhu J, Zhang L-J, Mitsumori NM. 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),* Vancouver, Canada. IEEE Computer Society, May 2009; 106–114.
23. Ma C, Du C, Zhang T, Hu F, Cai X. WSDL-based automated test data generation for Web service. In *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering,* Wuhan, China. IEEE Computer Society: Washington, DC, USA, December 2008; 731–737.
24. Offutt J, Xu W. Generating test cases for Web services using data perturbation. *ACM SIGSOFT Software Engineering Notes* 2004; **29**(5):1–10.
25. Sneed HM, Huang S. WSDLTest - a tool for testing Web services. In *WSE '06: Proceedings of the Eighth IEEE International Symposium on Web Site Evolution,* Philadelphia, PA, USA. IEEE Computer Society: Washington, DC, USA, September 2006; 14–21.
26. Chakrabarti SK, Kumar P. Test-the-REST: an approach to testing RESTful Web-services. In *ComputationWorld 2009 Future Computing Service Computation Cognitive Adaptive Content Patterns,* Athens, Grece. IEEE Computer Society, July 2009; 302–308.
27. Bai X, Lee S, Tsai WT, Chen Y. Ontology-based test modeling and partition testing of Web services. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services,* Beijing, China. IEEE Computer Society: Washington, DC, USA, September 2008; 465–472.
28. Dai G, Bai X, Wang Y, Dai F. Contract-based testing for Web services. In *COMPSAC 2007: Proceedings of the 31st Annual International Computer Software and Applications Conference,* Beijing, China. IEEE Computer Society: Washington, DC, USA, July 2007; **1**:517–526.
29. Tsai WT, Wei X, Chen Y, Paul R, Xiao B. Swiss cheese test case generation for Web services testing. *IEICE - Transactions on Information and Systems.* Oxford University Press: Oxford, UK, 2005; **E88-D**(12):2691–2698.
30. Wang Y, Bai X, Li J, Huang R. Ontology-based test case generation for testing Web services. In *ISADS '07: Proceedings of the Eighth International Symposium on Autonomous Decentralized Systems,* Sedona, AZ, USA. IEEE Computer Society: Washington, DC, USA, March 2007; 43–50.
31. Ye K, Huang J, Gong Y, Yang X. A static analysis method of WSDL related defect pattern in BPEL. In *ICCET 2010: Proceedings of the 2nd International Conference on Computer Engineering and Technology,* Chengdu, China. IEEE Computer Society, April 2010; V7–472–V7–475.
32. Bozkurt M, Harman M. Automatically generating realistic test input from Web services. In *SOSE '11: Proceedings of the 6th IEEE Symposium on Service-Oriented System Engineering,* Irvine, CA, USA. IEEE Computer Society, December 2011; 13–24.
33. Conroy KM, DC, Grechanik M, Hellige M, Liongosari ES, Xie Q. Automatic test generation from GUI applications for testing Web services. In *ICSM: Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM 2007),* Paris, France. IEEE Computer Society: Washington, DC, USA, October 2007; 345–354.
34. Bozkurt M, Harman M. Finding Test Data on the Web. Presented at the 2008 Testing: Academic & Industrial Conference (TAIC PART 2008), August 2008. Available from: http://www2008.taicpart.org/FastAbstracts/camera_ready/FastAbstract-11551.pdf [last accessed 27 March 2012].
35. Bozkurt M, Harman M. Optimised Realistic Test Input Generation. Presented at the SSBSE 2011: The 3rd International Symposium on Search Based Software Engineering, September 2011. Available from: http://www.ssbse.org/2011/fastabstracts/bozkurt.pdf [last accessed 27 March 2012].
36. Meyer B. Applying 'Design by Contract'. *Computer* 1992; **25**(10):40–51.
37. Jiang Y, Hou S-S, Shan J-H, Zhang L, Xie B. Contract-based mutation for testing components. In *ICSM: 21st IEEE International Conference on Software Maintenance (ICSM'05),* Budapest, Hungary. IEEE Computer Society, September 2005; 483–492.

DOI: 10.1002/stvr

38. Li ZJ, Maibaum T. An approach to integration testing of object-oriented programs. In *QSIC '07: Proceedings of the Seventh International Conference on Quality Software,* Portland, OR, USA. IEEE Computer Society: Washington, DC, USA, October 2007; 268–273.
39. Liang D, Xu K. Testing scenario implementation with behavior contracts. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference,* Chicago, IL, USA. IEEE Computer Society: Washington, DC, USA, September 2006; **1**:395–402.
40. Nebut C, Fleurey F, Le Traon Y, Jézéquel J-M. Requirements by contracts allow automated system testing. In *ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering,* Denver, CO, USA. IEEE Computer Society: Washington, DC, USA, November 2003; 85–96.
41. Heckel R, Lohmann M. Towards contract-based testing of Web services. *Proceedings of the International Workshop on Test and Analysis of Component Based Systems (TACoS 2004)*, Barcelona, Spain, March 2005; **116**:145–156.
42. Atkinson C, Barth F, Brenner D, Schumacher M. Testing Web-services using test sheets. In *ICSEA 2010: Proceedings of the 5th International Conference on Software Engineering Advances,* Nice, France. IEEE Computer Society, August 2010; 429–434.
43. Atkinson C, Brenner D, Falcone G, Juhasz M. Specifying high-assurance services. *Computer* 2008; **41**(8):64–71.
44. Tsai WT, Paul R, Wang Y, Fan C, Wang D. Extending WSDL to facilitate Web services testing. In *HASE'02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering,* Beijing, China. IEEE Computer Society: Washington, DC, USA, October 2002; 171–172.
45. Mei H, Zhang L. 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,* Beijing, China. IEEE Computer Society: Washington, DC, USA, October 2005; 199–206.
46. Noikajana S, Suwannasart T. An improved test case generation method for Web service testing from WSDL-S and OCL with pair-wise testing technique. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01,* Seattle, WA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 115–123.
47. Askarunisa A, Abirami AM, MadhanMohan S. A test case reduction method for semantic based Web services. In *2010 Second International Conference on Computing Communication and Networking Technologies,* Karur, India. IEEE, July 2010; 1–7.
48. Liu J, Lu X, Feng X, Liu J. OCL-based testing for e-learning Web service. In *ICWL'10: Proceedings of the 9th International Conference on New Horizons in Web-based Learning,* Shanghai, China. Springer-Verlag: Berlin, Heidelberg, December 2010; 161–168.
49. Mani S, Sinha VS, Sinha S, Dhoolia P, Mukherjee D, Chakraborty S. Efficient testing of service-oriented applications using semantic service stubs. In *ICWS '09: Proceedings of the 7th IEEE International Conference on Web Services,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 197–204.
50. Saleh I, Kulczycki G, Blake MB. Formal specification and verification of data-centric service composition. In *ICWS '10: Proceeedings of the 8th International Conference on Web Services,* Miami, FL, USA. IEEE Computer Society, July 2010; 131–138.
51. Weyuker E, Jeng B. Analyzing partition testing strategies. *IEEE Transactions on Software Engineering* 1991; **17**(7):703–711.
52. Heckel R, Mariani L. Automatic conformance testing of Web services. In *FASE 2005: Proceedings of the Fundamental Approaches to Software Engineering,* Edinburgh, Scotland. Springer, April 2005; 34–48.
53. Park Y, Jung W, Lee B, Wu C. Automatic discovery of Web services based on dynamic black-box testing. In *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01,* Seattle, WA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 107–114.
54. SOATest. Available from: http://www.parasoft.com/jsp/products/soatest.jsp?itemId=101 [last accessed 27 March 2012].
55. SOAP Sonar. Available from: http://www.crosschecknet.com/products/soapsonar.php [last accessed 27 March 2012].
56. HP Service Test. Available from: http://h71028.www7.hp.com/enterprise/cache/19054-0-0-225-121.html [last accessed 27 March 2012].
57. Oracle Application Testing Suite. Available from: www.oracle.com/technetwork/oem/app-quality-mgmt/application-quality-management-092933.html. [last accessed 27 March 2012].
58. Lenz C, Chimiak-Opoka J, Breu R. Model driven testing of SOA-based software. In *Proceedings of the Workshop on Software Engineering Methods for Service-Oriented Architecture (SEMSOA 2007),* Hannover, Germany, Lübke D (ed.). Leibniz Universität Hannover, FG Software Engineering, May 2007; 99–110.
59. Zhang YZ, Fu W, Qian JY. Automatic testing of Web services in Haskell Platform. *Journal of Computational Information Systems* 2010; **6**(9):2859–2867.
60. Shukla R, Carrington D, Strooper P. A passive test oracle using a component's API. In *APSEC '05: Proceedings of the 12th Asia-Pacific Software Engineering Conference,* Tapei, Taiwan. IEEE Computer Society: Washington, DC, USA, December 2005; 561–567.
61. Tsai WT, Chen Y, Cao Z, Bai X, Huang H, Paul RA. Testing Web services using progressive group testing. In *Advanced Workshop on Content Computing (AWCC 2004),* Zhenjiang, Jiangsu, China, Chi C-H, Lam K-Y (eds), Lecture Notes in Computer Science. Springer: China, 2004; **3309**:314–322.

DOI: 10.1002/stvr

62. Chan WK, Cheung SC, Leung KRPH. Towards a metamorphic testing methodology for service-oriented software applications. In *Proceedings of the 5th International Conference on Quality Software (QSIC 2005),* Melbourne, Australia. IEEE Computer Society: Washington, DC, USA, September 2005; 470–476.
63. Chen TY, Cheung SC, Yiu SM. 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, 1998.
64. FIT: Framework for Integrated Test. Available from: http://fit.c2.com/ [last accessed 27 March 2012].
65. Tsai WT, Chen Y, Zhang D, Huang H. 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),* Columbus, OH, USA. IEEE Computer Society: Washington, DC, USA, June 2005; 65–71.
66. Tsai WT, Chen Y, Paul R, Huang H, Zhou X, Wei X. 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,* Edinburgh, UK. IEEE Computer Society: Washington, DC, USA, July 2005; **2**:101–106.
67. Yue Q, Lu X, Shan Z, Xu Z, Yu H, Zha L. A model of message-based debugging facilities for Web or grid services. In *Proceedings of the 2009 Congress on Services - I,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 155–162.
68. Yue Q, Xu Z, Yu H, Li W, Zha L. An approach to debugging grid or Web services. In *ICWS'07: Proceedings of the 2007 IEEE International Conference on Web Services,* Salt Lake City, UT, USA. IEEE Computer Society: Los Alamitos, CA, USA, July 2007; 330–337.
69. Mayer P, Lübke D. 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,* Portland, Maine, USA. ACM: New York, NY, USA, 2006; 33–42.
70. Web Services Addressing (WS-Addressing). Available from: http://www.w3.org/Submission/ws-address1ing/ [last accessed 27 March 2012].
71. Li Z, Sun W, Jiang ZB, Zhang X. BPEL4WS unit testing: framework and implementation. In *ICWS '05: Proceedings of the 2005 IEEE International Conference on Web Services,* Orlando, FL, USA. IEEE Computer Society: Washington, DC, USA, July 2005; **1**:103–110.
72. Palomo-Duarte M, García-Domínguez A, Medina-Bulo I, Álvarez Ayllón A, Santacruz J. Takuan: a tool for WS-BPEL composition testing using dynamic invariant generation. In *ICWE'10: Proceedings of the 10th International Conference on Web Engineering*, Vienna, Austria, July 2010; 531–534.
73. Ilieva S, Pavlov V, Manova I. A composable framework for test automation of service-based applications. In *QUATIC '10: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology,* Oporto, Portugal. IEEE Computer Society: Washington, DC, USA, 2010; 286–291.
74. Reza H, Van Gilst D. A framework for testing RESTful Web services. In *ITNG '10: Proceedings of the 7th International Conference on Information Technology: New Generations,* Las Vegas, NV, USA. IEEE Computer Society: Washington, DC, USA, April 2010; 216–221.
75. Li X, Huai J, Liu X, Zeng J, Huang Z. SOArMetrics: a toolkit for testing and evaluating SOA middleware. In *Proceedings of the 2009 Congress on Services - I,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, 2009; 163–170.
76. Zhou ZQ, Huang DH, Tse TH, Yang Z, Huang H, Chen TY. Metamorphic testing and its applications. In *Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004)*, Xian, China, October 2004.
77. Morell LJ. A theory of fault-based testing. *IEEE Transactions on Software Engineering.* IEEE Press: Piscataway, NJ, USA. 1990; **16**(8):844–857.
78. Xu W, Offutt J, Luo J. Testing Web services by XML perturbation. In *ISSRE '05: Proceedings of the 16th IEEE International Symposium on Software Reliability,* Engineering *Chicago, IL, USA*. IEEE Computer Society: Washington, DC, USA, 2005; 257–266.
79. de Almeida LFJ, Vergilio SR. Exploring perturbation based testing for Web services. In *ICWS '06: Proceedings of the 2006 IEEE International Conference on Web Services,* Chicago, IL, USA. IEEE Computer Society: Washington, DC, USA, 2006; 717–726.
80. Zhang J, Zhang L-J. 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,* Orlando, FL, USA. IEEE Computer Society: Washington, DC, USA, July 2005; 621–628.
81. Vieira M, Laranjeiro N, Madeira H. Benchmarking the robustness of Web services. In *PRDC '07: Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing,* Melbourne, Victoria, Australia. IEEE Computer Society: Washington, DC, USA, December 2007; 322–329.
82. Martin E, Basu S, Xie T. Automated testing and response analysis of Web services. In *ICWS '07: Proceedings of the 2007 IEEE International Conference on Web Services,* Salt Lake City, UT, USA. IEEE Computer Society, July 2007; 647–654.
83. Salva S, Rabhi I. Stateful Web service robustness. In *ICIW '10: Proceedings of the 5th International Conference on Internet and Web Applications and Services,* Barcelona, Spain. IEEE Computer Society: Washington, DC, USA, May 2010; 167–173.
84. Wang Y, Ishikawa F, Honiden S. Business semantics centric reliability testing for Web services in BPEL. In *SERVICES '10: Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2010; 237–244.

DOI: 10.1002/stvr

85. Looker N, Xu J, Munro M. Determining the dependability of service-oriented architectures. *International Journal of Simulation and Process Modelling* 2007; **3**(26):88–97.
86. Juszczyk L, Dustdar S. Script-based generation of dynamic testbeds for SOA. In *ICWS '10: Proceedings of the 2010 IEEE International Conference on Web Services,* Miami, FL, USA. IEEE Computer Society, July 2010; 195–202.
87. Juszczyk L, Dustdar S. Testbeds for emulating dependability issues of mobile Web services. In *SERVICES '10: Proceedings of the 6th World Congress on Services,* Miami, FL, USA. IEEE Computer Society: Washington, DC, USA, July 2010; 683–686.
88. Siblini R, Mansour N. Testing Web services. In *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications,* Cairo, Egypt. IEEE Computer Society: Washington, DC, USA, January 2005; 135.
89. Lee S, Bai X, Chen Y. Automatic mutation testing and simulation on OWL-S specified Web services. In *ANSS-41 '08: Proceedings of the 41st Annual Simulation Symposium,* Ottawa, Canada. IEEE Computer Society: Washington, DC, USA, April 2008; 149–156.
90. Wang R, Huang N. Requirement model-based mutation testing for Web service. In *NWESP '08: Proceedings of the 2008 4th International Conference on Next Generation Web Services Practices,* Seoul, Korea. IEEE Computer Society: Washington, DC, USA, October 2008; 71–76.
91. Wang X, Huang N, Wang R. Mutation test based on OWL-S requirement model. In *ICWS '09: Proceedings of the 7th IEEE International Conference on Web Services,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 1006–1007.
92. SWRL: A Semantic Web Rule Language. Available from: http://www.w3.org/Submission/SWRL/ [last accessed 27 March 2012].
93. Apilli BS. Fault-based combinatorial testing of web services. In *OOPSLA '09: Proceeding of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications,* Orlando, Florida, USA. ACM: New York, NY, USA, October 2009; 731–732.
94. Watkins KZ. Introducing fault-based combinatorial testing to Web services. In *Proceedings of the IEEE SoutheastCon 2010,* Charlotte-Concord, NC, USA. IEEE Computer Society, March 2010; 131–134.
95. Fu C, Ryder BG, Milanova A, Wonnacott D. Testing of java web services for robustness. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis,* Boston, Massachusetts, USA. ACM, July 2004; 23–34.
96. Laranjeiro N, Canelas S, Vieira M. wsrbench: an on-line tool for robustness benchmarking. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing,* Honolulu, HI, USA. IEEE Computer Society: Washington, DC, USA, 2008; 187–194.
97. Laranjeiro N, Vieira M, Madeira H. Improving Web services robustness. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 397–404.
98. wsrbench. Available from: http://wsrbench.dei.uc.pt/ [last accessed 27 March 2012].
99. Laranjeiro N, Oliveira R, Vieira M. Applying text classification algorithms in Web services robustness testing. In *SRDS 2010: 29th IEEE Symposium on Reliable Distributed Systems,* New Delhi, India. IEEE Computer Society, November 2010; 255–264.
100. Bessayah F, Cavalli A, Maja W, Martins E, Valenti A. A fault injection tool for testing Web services composition. In *Testing – Practice and Research Techniques*, Vol. 6303, Bottaci L, Fraser G (eds), Lecture Notes in Computer Science. Springer: Berlin / Heidelberg, 2010; 137–146.
101. Domínguez-Jiménez JJ, Estero-Botaro A, García-Domínguez A, Medina-Bulo I. Gamera: an automatic mutant generation system for WS-BPEL compositions. In *Proceedings of the 7th European Conference on Web Services (ECOWS'09),* Eindhoven, Netherlands. IEEE Computer Society, November 2009; 97–106.
102. El-Far IK. Enjoying the perks of model-based testing. In *STARWEST 2001: Proceedings of the Software Testing, Analysis, and Review Conference*, San Jose, CA, USA, October 2001.
103. Yang Y, Tan Q, Xiao Y. 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,* Bremen, Germany. ACM: New York, NY, USA, November 2005; 47–54.
104. Morimoto S. A survey of formal verification for business process modeling. In *ICCS '08: Proceedings of the 8th International Conference on Computational Science, Part II,* Kraków, Poland. Springer-Verlag: Berlin, Heidelberg, June 2008; 514–522.
105. Endo AT, Simão AS, Souza SRS, Souza PSL. 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,* Windsor, UK. IEEE Computer Society: Washington, DC, USA, August 2008; 3–12.
106. Yan J, Li Z, Yuan Y, Sun W, Zhang J. 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,* Raleigh, NC, USA. IEEE Computer Society: Washington, DC, USA, November 2006; 75–84.
107. Yuan Y, Li Z, Sun W. A graph-search based approach to BPEL4WS test generation. In *ICSEA '06: Proceedings of the International Conference on Software Engineering Advances,* Tahiti, French Polynesia. IEEE Computer Society: Washington, DC, USA, October 2006; 14.

DOI: 10.1002/stvr

108. Lallali M, Zaidi F, Cavalli A. 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,* Shanghai, China. IEEE Computer Society: Washington, DC, USA, December 2007; 417–426.
109. Lallali M, Zaidi F, Cavalli A, Hwang I. Automatic timed test case generation for Web services composition. In *ECOWS '08: Proceedings of the 2008 6th European Conference on Web Services,* Dublin, Ireland. IEEE Computer Society: Washington, DC, USA, November 2008; 53–62.
110. Cao T-D, Felix P, Castanet R, Berrada I. Testing Web services composition using the TGSE tool. In *Proceedings of the 2009 Congress on Services - I,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 187–194.
111. Cao T-D, Felix P, Castanet R, Berrada I. Online testing framework for Web services. In *ICST '10: Proceedings of the 3rd International Conference on Software Testing, Verification and Validation,* Paris, France. IEEE Computer Society: Washington, DC, USA, April 2010; 363–372.
112. Li L, Chou W. A combinatorial approach to multi-session testing of stateful Web services. In *Proceedings of the 2009 Congress on Services - I,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 179–186.
113. Li L, Chou W. An abstract GFSM model for optimal and incremental conformance testing of Web services. In *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 205–212.
114. Belli F, Linschulte M. Event-driven modeling and testing of real-time web services. *Service Oriented Computing and Applications* 2010; **4**:3–15.
115. Endo AT, Linschulte M, Simão AS, Souza SRS. Event- and coverage-based testing of Web services. In *Proceedings of the 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion,* Singapore, Singapore, SSIRI-C '10. IEEE Computer Society: Washington, DC, USA, April 2010; 62–69.
116. Hou SS, Zhang L, Lan Q, Mei H, Sun JS. Generating effective test sequences for BPEL testing. In *QSIC 2009: Proceedings of the 9th International Conference on Quality Software*. IEEE Computer Society Press: Jeju, Korea, August 2009.
117. Paradkar AM, Sinha A, Williams C, Johnson RD, Outterson S, Shriver C, Liang C. Automated functional conformance test generation for semantic Web services. In *ICWS '07: Proceedings of the 2007 IEEE International Conference on Web Services*, Salt Lake City, UT, USA, July 2007; 110–117.
118. Guangquan Z, Mei R, Jun Z. 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,* Nanchang, China. IEEE Computer Society, December 2007; 59–65.
119. Pilone D, Pitman N. *UML 2.0 in a Nutshell (In a Nutshell (O'Reilly))*. O'Reilly Media, Inc: Sebastopol, CA, USA, 2005.
120. Ma C, Wu J, Zhang T, Zhang Y, Cai X. Testing BPEL with Stream X-Machine. In *ISISE '08: Proceedings of the 2008 International Symposium on Information Science and Engineering,* Shanghai, China. IEEE Computer Society: Washington, DC, USA, December 2008; 578–582.
121. Laycock G. The theory and practice of specification based software testing. *Ph.D. Dissertation*, University of Sheffield, 2003.
122. Casado R, Tuya J, Younas M. Testing long-lived Web services transactions using a risk-based approach. In *QSIC '10: Proceedings of the 10th International Conference on Quality Software,* Zhangjiajie, China. IEEE Computer Society: Washington, DC, USA, July 2010; 337–340.
123. Ali S, Briand LC, Hemmati H, Panesar-Walawege RK. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Transactions on Software Engineering* 2010; **36**:742–762.
124. Harman M. Automated test data generation using search based software engineering. In *AST '07: Proceedings of the Second International Workshop on Automation of Software Test,* Minneapolis, MN, USA. IEEE Computer Society: Washington, DC, USA, May 2007; 2.
125. McMinn P. Search-based software test data generation: a survey. *Software Testing, Verification & Reliability (STVR)* 2004; **14**(2):105–156.
126. Blanco R, García-Fanjul J, Tuya J. 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,* Denver, CO, USA. IEEE Computer Society: Washington, DC, USA, 2009; 131–140.
127. Tarhini A, Fouchal H, Mansour N. Regression testing Web services-based applications. In *Proceedings of the 4th ACS/IEEE International Conference on Computer Systems and Applications,* Sharjah, UAE. IEEE Computer Society: Washington, DC, USA, March 2006; 163–170.
128. Tarhini A, Fouchal H, Mansour N. A simple approach for testing Web service based applications. In *Proceedings of the 5th International Workshop on Innovative Internet Community Systems (IICS 2005),* Paris, France, Bui A, Bui M, Böhme T, Unger H (eds), Lecture Notes in Computer Science. Springer, 2005; **3908**:134–146.
129. King JC. Symbolic execution and program testing. *Communications of the ACM* 1976; **19**(7):385–394.
130. Legeard B. BZ-Testing-Tools: model-based test generator. *The 18th IEEE International Conference on Automated Software Engineering (ASE 2003) - Demo Paper*, Montreal, Canada, October 2003.

DOI: 10.1002/stvr

131. Sinha A, Paradkar A. 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,* Portland, Maine. ACM: New York, NY, USA, 2006; 17–22.
132. Disjunctive Normal Form. Available from: http://mathworld.wolfram.com/DisjunctiveNormalForm.html [last accessed 27 March 2012].
133. Bentakouk L, Poizat P, Zaïdi F. 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, Lecture Notes in Computer Science. Springer: Berlin / Heidelberg, November 2009; 16–32.
134. Zhou L, Ping J, Xiao H, Wang Z, Pu G, Ding Z. Automatically testing Web services choreography with assertions. In *ICFEM'10: Proceedings of the 12th International Conference on Formal Engineering Methods and Software Engineering,* Shanghai, China. Springer-Verlag: Berlin, Heidelberg, November 2010; 138–154.
135. Escobedo JP, Gaston C, Le Gall P, Cavalli A. Testing Web service orchestrators in context: a symbolic approach. In *SEFM '10: Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Methods,* Pisa, Italy. IEEE Computer Society: Washington, DC, USA, September 2010; 257–267.
136. Clarke EM, Schlingloff B-H. Model checking. In *Handbook of Automated Reasoning*, Vol. 2, ch. 24, Robinson JA, Voronkov A (eds). Elsevier Science Publishers B. V: Amsterdam, The Netherlands, 2001; 1635–1790.
137. Carroll J, Long D. *Theory of finite automata with an introduction to formal languages*. Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1989.
138. Gill A. *Introduction to the Theory of Finite-State Machines*. McGraw-Hill, 1962.
139. SPIN. Available from: http://spinroot.com/spin/whatispin.html [last accessed 27 March 2012].
140. NuSMV. Available from: http://nusmv.irst.itc.it/ [last accessed 27 March 2012].
141. BLAST. Available from: http://mtc.epfl.ch/software-tools/blast/ [last accessed 27 March 2012].
142. Fu X, Bultan T, Su J. Analysis of interacting BPEL Web services. In *WWW '04: Proceedings of the 13th International Conference on World Wide Web,* New York, New York, USA. ACM, May 2004; 621–630.
143. XML Path Language (XPath). Available from: http://www.w3.org/TR/xpath/ [last accessed 27 March 2012].
144. Promela Manual. Available from: http://spinroot.com/spin/Man/promela.html [last accessed on 27 March 2012].
145. Banieqbal B, Barringer H, Pnueli A (eds). *Temporal Logic in Specification, Altrincham, UK, April 8–10, 1987, Proceedings*, Lecture Notes in Computer Science, Vol. 398. Springer, 1989.
146. García-Fanjul J, de la Riva C, Tuya J. Generating test cases specifications for compositions of Web services. In *Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006),* Palermo, Italy, Bertolino A, Polini A (eds), June 2006; 83–94.
147. Zheng Y, Zhou J, Krause P. A model checking based test case generation framework for Web services. In *ITNG '07: Proceedings of the International Conference on Information Technology,* Las Vegas, NV, USA. IEEE Computer Society: Washington, DC, USA, April 2007; 715–722.
148. Zheng Y, Zhou J, Krause P. Analysis of BPEL data dependencies. In *EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications,* Lbeck, Germany. IEEE Computer Society: Washington, DC, USA, August 2007; 351–358.
149. Huang H, Tsai W-T, Paul RA, Chen Y. Automated model checking and testing for composite Web services. In *ISORC: 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005),* Seattle, WA, USA. IEEE Computer Society, May 2005; 300–307.
150. Jokhio MS, Dobbie G, Sun J. A framework for testing semantic Web services using model checking. In *SEEFM '09: Proceedings of the 4th South-East European Workshop on Formal Methods,* Thessaloniki, Grece. IEEE Computer Society: Washington, DC, USA, December 2009; 17–24.
151. Qi Z, Liu L, Zhang F, Guan H, Wang H, Chen Y. FLTL-MC: online high level program analysis for Web services. In *Proceedings of the 2009 Congress on Services - I,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 171–178.
152. Betin-Can A, Bultan T. Verifiable Web services with hierarchical interfaces. In *ICWS '05: Proceedings of the 2005 IEEE International Conference on Web Services,* Orlando, Florida, USA. IEEE Computer Society: Washington, DC, USA, July 2005; **1**:85–94.
153. Java PathFinder. Available from: http://javapathfinder.sourceforge.net/ [last accessed 27 March 2012].
154. Ramsokul P, Sowmya A. 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,* Pune, India. IEEE Computer Society: Washington, DC, USA, September 2006; 196–205.
155. Guermouche N, Godart C. Timed model checking based approach for Web services analysis. In *ICWS '09: Proceedings of the 7th IEEE International Conference on Web Services,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 213–221.
156. Yuan M, Huang Z, Li X, Yan Y. Towards a formal verification approach for business process coordination. In *ICWS '10: Proceedings of the 2010 IEEE International Conference on Web Services,* Miami, FL, USA. IEEE Computer Society: Washington, DC, USA, 2010; 361–368.
157. Murata T. Petri nets: properties, analysis and applications. In *Proceedings of the IEEE* 1989; **77**(4):541–580.
158. Dong W-L, Yu H. Web service testing method based on fault-coverage. In *EDOC Workshops: The 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06),* Hong Kong, China. IEEE Computer Society: Washington, DC, USA, October 2006; 43.

DOI: 10.1002/stvr

159. Ouyang C, Verbeek HMW, van der Aalst WMP, Breutel S, Dumas M, ter Hofstede AHM. WofBPEL: A Tool for Automated Analysis of BPEL Processes. In *Service-Oriented Computing - ICSOC 2005*. Springer-Verlag: Berlin / Heidelberg, 2005; **3826**:484–489.
160. WofBPEL and BPEL2PNML. Available from: http://www.bpm.fit.qut.edu.au/projects/babel/tools/ [last accessed 27 March 2012].
161. Schlingloff H, Martens A, Schmidt K. Modeling and model checking Web services. *Electronic Notes in Theoretical Computer Science* 2005; **126**:3–26.
162. Schmidt K. LoLA: A Low Level Analyser. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000),* Aarhus, Denmark, Vol. 1825/2000, Lecture Notes in Computer Science. Springer: Berlin / Heidelberg, June 2000; 465–474.
163. Lohmann N, Massuthe P, Stahl C, Weinberg D. Analyzing interacting BPEL processes. In *Business Process Management*, Vol. 4102/2006, Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, October 2006; 17–32.
164. Moser S, Martens A, Grlach K, Amme W, Godlinski A. Advanced verification of distributed WS-BPEL business processes incorporating CSSA-based data flow analysis. In *Proceedings of the 2007 IEEE International Conference on Services Computing (SCC 2007),* Salt Lake City, Utah, USA. IEEE Computer Society, July 2007; 98 –105.
165. Jensen K. Coloured Petri nets: status and outlook. In *Applications and Theory of Petri Nets 2003*, Vol. 2679, van der Aalst W, Best E (eds), Lecture Notes in Computer Science. Springer: Berlin / Heidelberg, 2003; 1–2.
166. CPNTOOLS. Available from: http://wiki.daimi.au.dk/cpntools/cpntools.wiki [last accessed 27 March 2012].
167. Yi X, Kochut KJ. 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,* San Diego, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2004; 756–760.
168. Dong W-L, Yu H, Zhang Y-B. Testing BPEL-based Web service composition using high-level Petri nets. In *EDOC'06: The 10th IEEE International Enterprise Distributed Object Computing Conference,* Hong Kong, China. IEEE Computer Society: Washington, DC, USA, October 2006; 441–444.
169. Dai G, Bai X, Zhao C. A framework for model checking Web service compositions based on BPEL4WS. In *ICEBE 2007: Proceedings of the IEEE International Conference on e-Business Engineering,* Hong Kong*, China*. IEEE Computer Society: Washington, DC, USA, October 2007; 165–172.
170. Xu C, Wang H, Qu W. Modeling and verifying BPEL using synchronized net. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied Computing,* Fortaleza, Ceara, Brazil. ACM: New York, NY, USA, March 2008; 2358–2362.
171. Petri Net Markup Language (PNML). Available from: http://www2.informatik.hu-berlin.de/top/pnml/about.html [last accessed 27 March 2012].
172. Li L, Chou W, Guo W. Control flow analysis and coverage driven testing for Web services. In *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services,* Beijing, China. IEEE Computer Society: Washington, DC, USA, September 2008; 473–480.
173. Resource Description Framework (RDF). Available from: http://www.w3.org/RDF/ [last accessed 27 March 2012].
174. Yang X, Huang J, Gong Y. Defect analysis respecting dead path elimination in BPEL process. In *APSCC '10: Proceedings of the 2010 IEEE Asia-Pacific Services Computing Conference,* Hangzhou, China. IEEE Computer Society: Washington, DC, USA, December 2010; 315–321.
175. Felderer M, Breu R, Chimiak-Opoka J, Breu M, Schupp F. Concepts for model-based requirements testing of service oriented systems. In *Software Engineering, SE 2009*. IASTED: Innsbruck, Austria, March 2009; 152–157.
176. Felderer M, Zech P, Fiedler F, Breu R. A tool-based methodology for system testing of service-oriented systems. In *VALID '10: Proceedings of the 2nd International Conference on Advances in System Testing and Validation Lifecycle,* Nice, France. IEEE Computer Society: Washington, DC, USA, August 2010; 108–113.
177. Frantzen L, Las Nieves Huerta M, Kiss ZG, Wallet T. On-the-fly model-based testing of Web services with Jambition. *Web Services and Formal Methods*, Vol. 5387, Bruni R, Wolf K (eds), Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, September 2009; 143–157.
178. PLASTIC Framework. Available from: http://plastic.isti.cnr.it/wiki/tools [last accessed 27 March 2012].
179. Cavalli AR, Cao T-D, Mallouli W, Martins E, Sadovykh A, Salva S, Zaïdi F. WebMov: a dedicated framework for the modelling and testing of Web services composition. In *International Conference on Web Services,* Paris, France. IEEE Computer Society: Washington, DC, USA, April 2010; 377–384.
180. Web Service Interoperability Organisation (WS-I). Basic Profile 1.2, November 2010. Available from: http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile [last accessed 27 March 2012].
181. Senthil Kumar KM, Das AS, Padmanabhuni S. WS-I basic profile: a practitioner's view. In *ICWS '04: Proceedings of the 2004 IEEE International Conference on Web Services,* San Diego, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2004; 17–24.
182. Skonnard A. Web services and datasets. MSDN magazine, April 2003. Available from: http://msdn.microsoft.com/en-us/magazine/cc188755.aspx [last accessed 27 March 2012].
183. Narita M, Shimamura M, Iwasa K, Yamaguchi T. Interoperability verification for Web service based robot communication platforms. In *ROBIO 2007: Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetics,* Sanya, China. IEEE Computer Society: Washington, DC, USA, December 2007; 1029–1034.

DOI: 10.1002/stvr

184. Yu Y, Huang N, Ye M. Web services interoperability testing based on ontology. In *CIT '05: Proceedings of the Fifth International Conference on Computer and Information Technology,* Binghamton, NY, USA. IEEE Computer Society: Washington, DC, USA, September 2005; 1075–1079.
185. Bertolino A, Polini A. The audition framework for testing Web services interoperability. In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA),* Porto, Portugal. IEEE Computer Society: Washington, DC, USA, August 2005; 134–142.
186. Jess Rule Engine. Available from: http://www.jessrules.com/jess/index.shtml [last accessed 27 March 2012].
187. Smythe C. 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),* Palermo Italy, Bertolino A, Polini A (eds), 2006; 95–119.
188. Ramsokul P, Sowmya A. A sniffer based approach to WS protocols conformance checking. In *ISPDC '06: Proceedings of The Fifth International Symposium on Parallel and Distributed Computing,* Timisoara, Romania. IEEE Computer Society: Washington, DC, USA, July 2006; 58–65.
189. Web Services Atomic Transaction (WS-AtomicTransaction). Available from: http://schemas.xmlsoap.org/ws/2004/10/wsat/ [last accessed 27 March 2012].
190. Web Services Business Activity (WS-BusinessActivity). Available from: http://docs.oasis-open.org/ws-tx/wsba/2006/06 [last accessed 27 March 2012].
191. Cavalli A, Gervy C, Prokopenko S. New approaches for passive testing using an extended finite state machine specification. *Information and Software Technology* 2003; **45**(12):837–852.
192. Benharref A, Dssouli R, Serhani M, En-Nouaary A, Glitho R. New approach for EFSM-based passive testing of Web services. In *Testing of Software and Communicating Systems*, Vol. 4581, Petrenko A, Veanes M, Tretmans J, Grieskamp W (eds), Lecture Notes in Computer Science. Springer: Berlin/Heidelberg, 2007; 13–27.
193. Andrés C, Cambronero M, Núñez M. Passive testing of Web services. In *Web Services and Formal Methods*, Vol. 6551, Bravetti M, Bultan T (eds), Lecture Notes in Computer Science. Springer: Berlin/Heidelberg, 2011; 56–70.
194. Morales G, Maag S, Cavalli A. Timed extended invariants for the passive testing of Web services. In *ICWS '10: Proceeedings of the 8th International Conference on Web Services,* Miami, FL, USA. IEEE Computer Society, July 2010; 76–82.
195. Cao T-D, Phan-Quang T-T, Félix P, Castanet R. Automated runtime verification for Web services. In *ICWS '10: Proceeedings of the 8th International Conference on Web Services,* Miami, FL, USA. IEEE Computer Society, July 2010; 76–82.
196. Tsai WT, Paul R, Cao Z, Yu L, Saimi A. Verification of Web services using an enhanced UDDI server. *Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, Turku, Finland, January 2003; 131–138.
197. Benedetto C. SOA and Integration Testing: The End-to-End View, September 2006. Available from: http://soa.sys-con.com/node/275057 [last accessed 27 March 2012].
198. Tsai WT, Paul R, Song W, Cao Z. Coyote: an XML-based framework for Web services testing. In *HASE'02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering,* Tokyo, Japan. IEEE Computer Society: Washington, DC, USA, October 2002; 173.
199. Duvall P, Matyas S, Glover A. *Continuous integration: improving software quality and reducing risk*. Addison-Wesley Professional: Upper Saddle River, NJ, USA, 2007.
200. Huang HY, Liu HH, Li ZJ, Zhu J. Surrogate: a simulation apparatus for continuous integration testing in service oriented architecture. In *IEEE SCC: 2008 IEEE International Conference on Services Computing (SCC 2008),* Honolulu, Hawaii, USA, Vol. 2. IEEE Computer Society, July 2008; 223–230.
201. Liu H, Li Z, Zhu J, Tan H, Huang H. A unified test framework for continuous integration testing of SOA solutions. In *ICWS '09: Proceedings of the 7th IEEE International Conference on Web Services,* Los Angeles, CA, USA. IEEE Computer Society: Washington, DC, USA, July 2009; 880–887.
202. Peyton L, Stepien B, Seguin P. Integration testing of composite applications. In *HICSS '08: Proceedings of the 41st Annual Hawaii International Conference on System Sciences,* Waikoloa, Big Island, Hawaii. IEEE Computer Society: Washington, DC, USA, January 2008; 96.
203. ETSI ES 201 873-1. The Testing and Test Control Notation version 3, Part1: TTCN-3 Core notation, V2.1.1, June 2005. Available from: http://www.ttcn-3.org/StandardSuite.htm [last accessed 27 March 2012].
204. Mei L, Chan WK, Tse TH. Data flow testing of service-oriented workflow applications. In *ICSE '08: Proceedings of the 30th International Conference on Software Engineering,* Leipzig, Germany. ACM: New York, NY, USA, May 2008; 371–380.
205. de Angelis F, Polini A, de Angelis G. A counter-example testing approach for orchestrated services. In *ICST '10: Proceedings of the 3rd International Conference on Software Testing, Verification and Validation,* Paris, France. IEEE Computer Society: Washington, DC, USA, 2010; 373–382.
206. Yu Y, Huang N, Luo Q. 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,* Seoul, South Korea. IEEE Computer Society: Washington, DC, USA, October 2007; 31–34.
207. Tsai WT, Chen Y, Paul R, Liao N, Huang H. Cooperative and group testing in verification of dynamic composite Web services. In *COMPSAC '04: Proceedings of the 28th Annual International Computer Software and*

DOI: 10.1002/stvr

Applications Conference - Workshops and Fast Abstracts, Vol. 2, Hong Kong, China. IEEE Computer Society: Washington, DC, USA, September 2004; 170–173.
208. Bai X, Wang Y, Dai G, Tsai WT, Chen Y. 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*, Vol. 4608, Schmidt HW, Crnkovic I, Heineman GT, Stafford JA (eds), Lecture Notes in Computer Science. Springer: Medford, Massachusetts, USA, 2007; 258–273.
209. Zhang Y, Zhu H. Ontology for service oriented testing of Web services. In *SOSE '08: Proceedings of the 2008 IEEE International Symposium on Service-Oriented System Engineering,* Jhongli, Taiwan. IEEE Computer Society: Washington, DC, USA, 2008; 129–134.
210. Zhu H. A framework for service-oriented testing of Web services. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference,* Chicago, IL, USA, Vol. 2. IEEE Computer Society: Washington, DC, USA, September 2006; 145–150.
211. Bartolini C, Bertolino A, Elbaum S, Marchetti E. 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,* Amsterdam, The Netherlands. ACM: New York, NY, USA, August 2009; 161–170.
212. Eler MM, Delamaro ME, Maldonado JC, Masiero PC. Built-in structural testing of Web services. In *SBES '10: Proceedings of the 2010 Brazilian Symposium on Software Engineeringm,* Salvador, Brasil. IEEE Computer Society: Washington, DC, USA, October 2010; 70–79.
213. Campanella M, Chivalier P, Sevasti A, Simar N. *Quality of service definition*, Service Quality across Independently Managed Networks (SEQUIN), March 2001.
214. Web Services Glossary. Available from: http://www.w3.org/TR/ws-gloss/ [last accessed 27 March 2012].
215. Lee K, Jeon J, Lee W, Jeong S-H, Park S-W. QoS for Web services: requirements and possible approaches. *Technical Report*, W3C, November 2003. Available from http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/ [last accessed 27 March 2012].
216. Web Services Agreement Specification (WSAgreement). Available from: www.ogf.org/documents/GFD.107.pdf [last accessed 27 March 2012].
217. Chandrasekaran S, Miller J, Silver G, Arpinar IB, Sheth A. Composition, performance analysis and simulation of Web services. *Electronic Markets: The International Journal of Electronic Commerce and Business Media (EM)* June 2003; **13**(2):120–132.
218. Di Penta M, Canfora G, Esposito G, Mazza V, Bruno M. Search-based testing of service level agreements. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 2007),* London, England, UK. ACM, July 2007; 1090–1097.
219. Di Penta M, Bruno M, Esposito G, Mazza V, Canfora G. Web services regression testing. In *Test and Analysis of Web Services*, Baresi L, di Nitto E (eds). Springer: Berlin / Heidelberg, 2007; 205–234.
220. Gu Y, Ge Y. Search-based performance testing of applications with composite services. In *Proceedings of the 2009 International Conference on Web Information Systems and Mining (WISM 2009),* Shanghai, China. IEEE Computer Society, November 2009; 320–324.
221. Palacios M, García-Fanjul J, Tuya J, de la Riva C. A proactive approach to test service level agreements. In *ICSEA '10: Proceedings of the 5th International Conference on Software Engineering Advances,* Nice, France. IEEE Computer Society: Washington, DC, USA, August 2010; 453–458.
222. Bertolino A, De Angelis G, Polini A. Automatic generation of test-beds for pre-deployment QoS evaluation of Web services. In *Proceedings of the Sixth International Workshop on Software and Performance (WOSP 2007),* Buenos Aires, Argentina. ACM, February 2007; 137–140.
223. Bertolino A, De Angelis G, Frantzen L, Polini A. Model-based generation of testbeds for Web services. In *Proceedings of the 8th Testing of Communicating Systems and Formal Approaches to Software Testing (TESTCOM/FATES 2008),* Tokyo, Japan, Lecture Notes in Computer Science. Springer, 2008; 266–282.
224. Grundy J, Hosking J, Li L, Liu N. Performance engineering of service compositions. In *SOSE '06: Proceedings of the 2006 International Workshop on Service-Oriented Software Engineering,* Shanghai, China. ACM: New York, NY, USA, 2006; 26–32.
225. MaramaMTE. Available from: https://wiki.auckland.ac.nz/display/csidst/MaramaMTE [last accessed 27 March 2012].
226. Driss M, Jamoussi Y, Ghezala HHB. QoS testing of service-based applications. In *Proceedings of the 3rd IEEE International Design and Test Workshop (IDT '08),* Monastir, Tunisia. IEEE, December 2008; 45–50.
227. Pretre V, Bouquet F, Lang C. Using common criteria to assess quality of Web services. In *ICSTW '09: Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops,* Denver, CO, USA. IEEE Computer Society: Washington, DC, USA, 2009; 295–302.
228. Yeom G, Yun T, Min D. QoS model and testing mechanism for quality-driven web services selection. In *SEUS-WCCIA '06: Proceedings of the The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance,* Gyeongju, Korea. IEEE Computer Society: Washington, DC, USA, 2006; 199–204.
229. Yoo S, Harman M. Regression testing minimisation, selection and prioritisation: a survey. *Software Testing, Verification, and Reliability* 2010. DOI: 10.1002/stvr.430.

230. Rothermel G, Harrold MJ. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 1997; **6**(2):173–210.
231. Ruth M, Tu S. Concurrency issues in automating RTS for Web services. In *ICWS '07: Proceedings of the 2007 IEEE International Conference on Web Services,* Salt Lake City, UT, USA. IEEE Computer Society: Washington, DC, USA, July 2007; 1142–1143.
232. Ruth M, Tu S. 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,* Mauritius. IEEE Computer Society: Washington, DC, USA, May 2007; 47.
233. Liu H, Li Z, Zhu J, Tan H. Business Process Regression Testing. In *Proceedings of the 5th International Conference on Service-Oriented Computing,* Vienna, Austria. Springer-Verlag: Berlin Heidelberg, 2007; 157–168.
234. Mei L, Chan WK, Tse TH, Merkel RG. 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. IEEE Computer Society Press, August 2009; 21–30.
235. Tsai WT, Zhou X, Paul RA, Chen Y, Bai X. 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,* Dallas, TX, USA. IEEE Computer Society: Washington, DC, USA, November 2007; 105–112.
236. Hou S-S, Zhang L, Xie T, Sun J-S. Quota-constrained test-case prioritization for regression testing of service-centric systems. In *ICSM: 24th IEEE International Conference on Software Maintenance (ICSM 2008),* Beijing, China. IEEE, October 2008; 257–266.
237. Lin F, Ruth M, Tu S. Applying safe regression test selection techniques to Java Web services. In *NWESP '06: Proceedings of the International Conference on Next Generation Web Services Practices,* Seoul, South Korea. IEEE Computer Society: Washington, DC, USA, September 2006; 133–142.
238. Web Services Metadata Exchange (WS-MetadataExchange). Available from: http://www.w3.org/TR/2009/WD-ws-metadata-exchange-20090317/ [last accessed 27 March 2012].
239. Ruth M, Oh S, Loup A, Horton B, Gallet O, Mata M, Tu S. Towards automatic regression test selection for Web services. *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference*, Vol. 2, Beijing, China, July 2007; 729–736.
240. Harrold MJ, Jones JA, Li T, Liang D, Orso A, Pennings M, Sinha S, Spoon SA, Gujarathi A. Regression test selection for Java software. In *OOPSLA '01: Proceedings of the 16th ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications,* Tampa Bay, Florida, USA. ACM: New York, NY, USA, 2001; 312–326.
241. Apache Web Services Project - Apache Axis. Available from: http://ws.apache.org/axis/ [last accessed 27 March 2012].
242. WSDL2Java. Available from: http://cwiki.apache.org/CXF20DOC/wsdl-to-java.html [last accessed 27 March 2012].
243. Wang D, Li B, Cai J. Regression testing of composite service: an XBFG-based approach. In *Proceedings of the 2008 IEEE Congress on Services Part II (SERVICES-2 '08),* Beijing, China. IEEE Computer Society: Washington, DC, USA, 2008; 112–119.
244. Li B, Qiu D, Ji S, Wang D. Automatic test case selection and generation for regression testing of composite service based on extensible BPEL flow graph. In *ICSM '10: Proceedings of the 26th IEEE International Conference on Software Maintenance,* Timişoara, Romania. IEEE Computer Society: Washington, DC, USA, September 2010; 1–10.
245. Yang B, Wu J, Liu C, Xu L. A regression testing method for composite Web service. In *ICBECS 2010: 2010 International Conference on Biomedical Engineering and Computer Science,* Wuhan, China. IEEE Computer Society, April 2010; 1–4.
246. Mei L, Zhang Z, Chan WK, Tse TH. Test case prioritization for regression testing of service-oriented business applications. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, Madrid, Spain. ACM: New York, NY, USA, 2009; 901–910.
247. Athira B, Samuel P. Web services regression test case prioritization. In *CISIM 2010: Proceedings of the 2010 International Conference on Computer Information Systems and Industrial Management Applications,* Kraków, Poland. IEEE Computer Society, October 2010; 438–443.
248. Chen L, Wang Z, Xu L, Lu H, Xu B. Test case prioritization for Web service regression testing. In *SOSE 2010: Proceedings 5th IEEE International Symposium on Service Oriented System Engineering,* Loughborough, United Kingdom. IEEE Computer Society, June 2010; 173–178.
249. Zhai K, Jiang B, Chan WK, Tse TH. Taking advantage of service selection: a study on the testing of location-based Web services through test case prioritization. In *ICWS '10: Proceedings of the 8th IEEE International Conference on Web Services,* Miami, FL, USA. IEEE Computer Society: Washington, DC, USA, July 2010; 211–218.
250. Pautasso C. 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,* Dallas, TX, USA. IEEE Computer Society: Washington, DC, USA, September 2005; 311–313.
251. Pautasso C, Alonso G. The JOpera visual composition language. *Journal of Visual Languages and Computing (JVLC)* 2005; **16**(1-2):119–152.
252. Iskold A. Web 3.0: When Web Sites Become Web Services, March 2007. Available from: www.readwriteweb.com/archives/web_30_when_web_sites_become_web_services.php [last accessed 27 March 2012].

DOI: 10.1002/stvr

253. Rubel S. The Future is Web Services, Not Web Sites, March 2008. Available from: http://www.micropersuasion.com/2008/03/the-future-is-w.html [last accessed 05 August 2010].
254. Bhiri S, Gaaloul W, Rouached M, Hauswirth M. Semantic Web services for satisfying SOA requirements. In *Advances in Web Semantics I*, Vol. 4891, Dillon T, Chang E, Meersman R, Sycara K (eds), Lecture Notes in Computer Science. Springer: Berlin / Heidelberg, 2009; 374–395.
255. Fox J, Borenstein J. Semantic Discovery for Web Services, SOA World Magazine, March 2003. Available from: http://soa.sys-con.com/node/39718 [last accessed 27 March 2012].
256. Apple iTunes. Available from: http://www.apple.com/itunes/ [last accessed 27 March 2012].
257. Salva S, Rabhi I. A preliminary study on BPEL process testability. In *ICSTW '10: Proceedings of the 3rd International Conference on Software Testing, Verification, and Validation Workshops,* Paris, France. IEEE Computer Society: Washington, DC, USA, April 2010; 62–71.
258. Bucchiarone A, Melgratti H, Severoni F. Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE07)*, Mar del Plata, Argentina, 2007.
259. Mei L, Chan WK, Tse TH. Data flow testing of service choreography. 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,* Amsterdam, The Netherlands. ACM: New York, NY, USA, 2009; 151–160.
260. Stefanescu A, Wendland M-F, Wieczorek S. Using the UML testing profile for enterprise service choreographies. In *36th EUROMICRO Conference on Software Engineering and Advanced Application (SEAA),* Lille, France. IEEE Computer Society, 2010; 12–19.
261. Stefanescu A, Wieczorek S, Kirshin A. MBT4Chor: a model-based testing approach for service choreographies. In *Model Driven Architecture - Foundations and Applications*, Vol. 5562, Paige R, Hartman A, Rensink A (eds), Lecture Notes in Computer Science. Springer: Berlin / Heidelberg, 2009; 313–324.
262. Wieczorek S, Kozyura V, Roth A, Leuschel M, Bendisposto J, Plagge D, Schieferdecker I. Applying model checking to generate model-based integration tests from choreography models. In *TESTCOM '09/FATES '09: Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop,* Eindhoven, The Netherlands. Springer-Verlag: Berlin, Heidelberg, 2009; 179–194.

DOI: 10.1002/stvr