Automated Realistic Test Input Generation and Cost Reduction in Service-centric System Testing

Mustafa Bozkurt

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University of London.

Department of Computer Science

University College London

June 4, 2013

This thesis is dedicated to my grandmother,

Emine Aytaç (1924 - 2006) (اَلَّه يَرهَمهَ)

who wished to see me achieve this degree more than anyone.

Declaration

I, Mustafa Volkan Bozkurt confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Service-centric System Testing (ScST) is more challenging than testing traditional software due to the complexity of service technologies and the limitations that are imposed by the SOA environment. One of the most important problems in ScST is the problem of realistic test data generation. Realistic test data is often generated manually or using an existing source, thus it is hard to automate and laborious to generate. One of the limitations that makes ScST challenging is the cost associated with invoking services during testing process.

This thesis aims to provide solutions to the aforementioned problems, automated realistic input generation and cost reduction in ScST. To address automation in realistic test data generation, the concept of Service-centric Test Data Generation (ScTDG) is presented, in which existing services used as realistic data sources. ScTDG minimises the need for tester input and dependence on existing data sources by automatically generating service compositions that can generate the required test data. In experimental analysis, our approach achieved between 93% and 100% success rates in generating realistic data while state-of-the-art automated test data generation achieved only between 2% and 34%.

The thesis addresses cost concerns at test data generation level by enabling data source selection in ScTDG. Source selection in ScTDG has many dimensions such as cost, reliability and availability. This thesis formulates this problem as an optimisation problem and presents a multi-objective characterisation of service selection in ScTDG, aiming to reduce the cost of test data generation.

A cost-aware pareto optimal test suite minimisation approach addressing testing cost concerns during test execution is also presented. The approach adapts traditional multi-objective minimisation approaches to ScST domain by formulating ScST concerns, such as invocation cost and test case reliability. In experimental analysis, the approach achieved reductions between 69% and 98.6% in monetary cost of service invocations during testing.

Acknowledgements

First and foremost, I must thank Allah (سُبهَنَهُ وَ تَأَلَ)[1] to Whom we owe everything (اَلهَمدُ لِلَّه)[2]. All the good in this thesis is from Him (سُبهَنَهُ وَ تَأَلَ) and whatever mistakes herein is due to myself, as He (سُبهَنَهُ وَ تَأَلَ) said in the Qur'an "Whatever of good reaches you, is from Allah, but whatever of evil befalls you, is from yourself" [4:79].

Second, I am eternally grateful to the people mentioned here as this thesis might not have been completed without their help and guidance. I should thank my supervisor, Mark Harman, for being the best supervisor I could hope for. He did not just help me improve myself both intellectually and academically, but also showed incredible patience and understanding throughout the completion of this thesis. I am also thankful to my second supervisor, Nicolas Gold, for providing me with valuable criticism and for discussions that allowed me to realise other research areas where I can apply my ideas. I am also thankful to Youssef Hassoun for his help during the beginning of my PhD and his contributions towards the completion of the survey which constitutes a large part of Chapter 2. I am also thankful to William Langdon for providing me with invaluable constructive criticism and for patiently listening to my blunt questions. I also thank my colleagues in CREST (too many to mention) for all their help and support.

Lastly, I'm most grateful to my parents and my sister who did not just support me financially but also supported me emotionally. Without their help and encouragement, I am not sure if I would had been able to complete this thesis. I should especially thank my mother for believing in me my entire life even when I lost faith in myself from time to time.

[1] سُبهَنَهُ وَ تَأَلَ (Subhanahu wa ta'ala) means 'Glorified and exalted be He'.

[2] اَلهَمدُ لِلَّه (Al-hamdu lillah) means 'Praise be to the God'

Contents

List of Figures

List of Tables

Chapter 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 has brought about. The focus of businesses is migrating from product manufacturing (hardware and software) to service provision. For example, according to AppLabs [8], in the future, business concentration will shift away from system development towards core business. One motivation for this shift is the ability to build systems dynamically using services provided by other businesses. According to the Wintergreen Research Inc. (WRI) [355], the changes in the software industry go beyond technical issues. With software becoming more agile in order to fit business requirements, many businesses position themselves towards adopting Service-centric System(s) (ScS).

The market growth of SOC-related technology also supports this claim. WRI's estimation for the 2005 SOC market is $450 million and it is expected to reach $18.4 billion in 2012. According to the International Data Corporation [148], a leading market research institution, the global spend on service-oriented software in 2006 was nearly $2 billion. The same report projected that this spend will be $14 billion in 2011. A more recent report from Gartner [36] estimated Software as a Service (SaaS) and cloud-based business application services market as $13.4 billion in 2011 and projected that the market will grow to $32.2 billion in 2016.

However, when compared to optimistic predictions, SOC may be yet to achieve its full market potential. The author believes 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.

> *"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"* [64].

As stated by CDBI Forum [64], 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 systems raises the importance of establishing trust between the service provider and the consumer. The trust issue is not just about the 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 that, by nature, have the ability to dynamically select and use services. In order to confirm the correct functioning of ScS, interoperability among all its components, and integration of these components must be adequately tested.

ScS 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 a well known practice in other fields such as web hosting and network services. Due to 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 affect testing and monitoring for QoS in SOA.

As a result of the aforementioned issues, ScS may require more frequent testing than traditional software. The possibility of changes to an ScS (changes which requires testing) might be increased due to two major benefits of services: First, for ScS using services statically[1], this possibility increases with the number of services invoked. Second, the ease of integrating or removing a functionality (through a service) encourages developers to make additions which increases this possibility.

In testing of software systems, one common problem that many testers face is the problem of cost. The testing cost is often associated with the time it takes to complete the execution of a test suite (which may also include setting up the testing environment). However, with the introduction of services, this standard understanding of the term 'testing cost' has started to gain new dimensions, such as monetary cost of service invocation. Recent surveys [44, 49], as well as Chapter 2 of this thesis, identify runtime testing cost as one of the challenges of ScST. Testing cost is identified as a challenge due to three side effects that might be caused by the increased frequency of testing. These are:

1. The monetary cost of invoking service during runtime testing.
2. Service disruptions that might be caused by extensive testing.
3. Effects of testing in some systems, such as stock-exchange systems, where each usage of the service requires a business transaction, that carries with it a monetary cost.

There are also other challenges in ScST, such as identifying the right time to test

[1]Concepts of static and dynamic service use is explained in Chapter 2

the ScS and the number of test cases to run. This issue may occur due to different payment plans offered for the service invocations in SOA. For example, in the case of testing an ScS invoking services with quota-based payment plans, a test suite that can be executed within a given period while invoking a certain number of services may be needed in order to avoid extra charges.

There are existing approaches targeted at reducing these side effects, such as using mock/stub services or a testing environment where test cases are executed on a special instance of the Service/System Under Test (SUT). Although these approaches can reduce the severity of the mentioned side effects, they do not eliminate the need for runtime testing. In order to reduce the side effects of testing for ScS, approaches that reduce the testing cost at runtime are needed.

Another identified challenge in ScST is automated test data generation [44]. Even though many approaches for automated test data generation are proposed for traditional systems [166, 240, 259, 277] and ScS [16, 20, 29, 130, 185, 201, 235, 295], they are effective in generating test inputs only for certain input types. Their ineffectiveness in generating other input types is caused by the requirements of the inputs. This is because these inputs must be correctly formed and also contain semantics that tie them to real-world entities. We classify these type of inputs as 'realistic inputs'[2]. The ineffectiveness of the current automated approaches is also a factor in the testing cost. The extra cost is the human effort incurred in two ways: the tester's effort to verify automatically generated data, and the tester's effort to manually generate data. New automated solutions that can effectively generate realistic test cases are needed to address this issue.

The thesis of this dissertation is that existing web services can be used to increase the effectiveness of automated realistic test data generation and that multi-objective search algorithms can help reduce testing cost in ScST. The problem of effectiveness in automated generation of realistic inputs is addressed by the concept of service-centric test data generation. The cost reduction problem is addressed at two different levels:

[2]The concept of realistic data is explained in Chapter 4 in detail

test data generation and test execution. The cost reduction is achieved by formulating the problems at these two levels of abstraction as multi-objective optimisation problems. The approaches presented in this thesis are evaluated using real-world (where possible) and synthetic case studies (based on real-world parameters). The results from the experimental studies provide evidence that these approaches can be used in testing real-world ScS.

1.1 An Illustrative Scenario

This section describes a testing scenario explaining the cost involved at each stage and the need for realistic test data. Later part of the section also explain how this thesis address these problems.

First, we discuss the two possible scenarios that we consider a tester might be in SOA; testing a composition or testing a third-party service as a certifier[3]. In both of these cases, the tester might need to invoke third-party service(s) for testing purposes. Suppose that some of these invoked services require inputs such as International Standard Book Number (ISBN), Zone Improvement Plan (ZIP) code or Universal Product Code (UPC). Generating inputs automatically for these input types is not as straightforward as generating others, because in order to receive positive outputs from these services, generated test inputs must represent real-world entities. For example, a generated ISBN must represent an existing book, while a UPC must represent an existing product.

The tester might try some of the existing automated test data generators, however, he/she will soon realise that (as discussed in Chapter 4) these tools are not effective enough at generating ISBNs, ZIP codes and UPCs. When automated tools fail, a tester often regresses to one of the most practised methods: manual test data generation. The tester knows manual generation is effective, but it is also laborious and error-prone. In order to avoid errors and reduce effort, the tester might use some of the existing sources for generating the required test data. These sources might include:

[3]Testing perspectives in SOA are discussed in detail in Chapter 2

1. Existing data sources such as recorded user sessions and existing data bases.

2. Existing online sources such as websites like Amazon for ISBNs and the U.S.Postal Service for ZIP codes.

3. Domain knowledge to manually generate the data. For example, the tester might use ZIP codes or ISBNs he/she knows.

Even though these options seem to be viable for generating the required test data, the tester will soon realise that all of them fail at some level. For example, in order to use existing data sources the data has to be accessible to the tester, which might not be the case in every scenario. There is also the problem of temporal data. This is a type of data considered to be valid only for a certain period of time, such as flight details. If the required test data is temporal, some of the existing data such as recorded user sessions might not be reusable.

In traditional testing, it is often expected that the tester will execute each component with a number of positive and negative test cases. However, a small number of test cases might not be adequate in SOA due to the need for determining the Quality of Service (QoS) scores of a service[4]. This need occurs, for example, in the case of the certifier who needs to test and measure different aspects of a service such as performance, security and reliability. As a result, the tester might need to acquire a high number of test cases to perform adequate testing. However, the tester will soon realise that generating the required number of ISBNs and ZIP codes might take long time when they are to be generated manually.

The tester might also face the problem of *data limitation*, such as testing a service from a book publisher. In this case, the tester will know or realise that generating a valid ISBN that represents a real-world book is insufficient. What the tester needs as a positive test case is an ISBN that represents a book published by an appropriate publisher. This requirement is another factor that might make generation of the required test data more challenging.

[4]The concept of QoS and QoS testing for services are discussed in Chapter 2 and 5

Even though the tester generates a test suite that can achieve extensive testing, he/she might not be able to execute all test cases due to several limitations such as time constraints and testing budget. These limitations might be more severe in environments such as SOA, where testing frequency is higher. As a result, the tester will try to reduce the size of the test suite and will thus face the problem of test case selection. The tester has to find the most effective subset of the test suite that can be executed within the given time and budget. The testing budget is often considered to comprise of testers' wages, tool costs and other related expenses [162]. However, in ScST, there is also another cost that increases the overall cost of testing: the cost of service invocation. As a result, the tester must apply a cost-aware test suite minimisation.

1.2 Problems Addressed by this Thesis

This section describes the problems in the automation of test data generation for certain input types and cost reduction at two levels of testing process in ScST. The problems discussed in the section are real-world problems that a tester might face in ScST. However, the thesis does not claim that the provided solutions can be applicable to every scenario. The main aim of this thesis is to highlight the importance of these challenges, introduce new solutions and provide evidence to the feasibility and applicability of these new solutions.

1.2.1 Problem of Automated Realistic Test Data Generation

As discussed, generating test data for input types that represent a real-world entity is problematic. We refer to these inputs as 'realistic inputs'. Unfortunately, the existing automated tools are not effective enough to be used in generating data for these inputs and manual generation is laborious and error-prone. There is a need for automated tools that can effectively generate realistic test data.

This thesis presents a new technique called 'Service-centric Test Data Generation' (ScTDG) that can automate the generation of realistic test data by leveraging the data that can be acquired from existing services. ScTDG is effective in the automation realistic test data due to three main reasons:

1. The data provided by existing services are realistic by nature.

2. The dynamic nature of SOA which allows dynamic discovery and invocation of services.

3. The same set of services can be used in many cases to generate a large number of different test data.

ScTDG greatly benefits from the dynamic aspects of SOA: dynamic discovery and invocation. As a result of these benefits, it minimises the need for the tester input and reduce the dependency on previous data. Using ScTDG enables generation of test data based on the tester's specifications, thus helping in overcoming data limitation problems. ScTDG also helps with reducing the *human-oracle cost* due to the realism of the data generated from services as discussed in Chapter 4.

1.2.2 Problem of Cost Reduction in Test Data Generation

As mentioned, one of the main aims of a tester is to complete the testing process within the given testing budget. In a scenario where ScTDG is used, the tester will also need to consider the monetary cost of test data generation. This cost might arise due to the cost of invoked services during ScTDG. Since in ScTDG a tester can use any combination of suitable services, it is safe to assume he/she will want to use the services with lower costs in order to minimise testing cost.

However, cost might not be the only concern that the tester has in ScTDG. There are other concerns such as the reliability of the test data source (the service(s)) and the time it takes to generate the required data that the tester might be cautious of. Because the services used in ScTDG are third-party, it is safe to assume that, in many cases, the tester will want to use highly reliable services to generate the required test data. He/she might also want to use services with low response time or high availability in some scenarios.

This thesis considers the service selection problem in ScTDG as a multi-objective optimisation problem and introduces a pareto-optimal test data generation extension to

ScTDG. The application of optimisation techniques (both single- and multi-objective) have been proposed before in the field of test data generation [132]. However, none of the previously proposed approaches consider SOA concerns. The proposed optimisation benefits the existence of pre-measured QoS attributes of services and helps the tester to make an informed decision on which services to use in ScTDG. The multi-objective formulation also enables the tester to make trade-offs between various QoS aspects as discussed in Chapter 5.

1.2.3 Problem of Cost Reduction in Runtime Testing

One of the limitations in ScST is the cost associated with testing. The side effects of this limitation might be especially severe in service compositions with a large number of services. Even though there is existing work that aims to reduce the severity of this problem such as simulated testing [146, 186, 205, 208, 247, 270], these approaches do not eliminate the need for runtime testing with real services.

Test suite minimisation is one of the oldest methods aimed at reducing testing cost [374]. The aim of minimisation is reducing the testing cost through redundant test case elimination. The existing cost-aware test case prioritisation and minimisation approaches often associate cost with the execution time of test suites. However, reduction in the test cases brings up other concerns such as preserving branch coverage and fault detection capability. Unfortunately, none of the existing work considers ScST concerns such as cost of service invocations and test data reliability (based on its source).

This thesis combines the concerns of the ScST domain with some of the traditional concerns and formulates them all as optimisation problems. The application of multi-objective algorithms to these problems enables the tester to discover low cost subsets of the test suite, while preserving achievement of test aims such as branch coverage. Pareto-optimal test suite minimisation also enables the tester to make trade-offs between the concerns.

1.3 Contributions of this Thesis

The contributions of this thesis are as follows:

1. The analysis of the existing and future trends in ScST, which helps build a roadmap to its future.

2. The automation of realistic test data generation with ScTDG and the empirical evaluation of the concept using real-world scenarios, which provides evidence for its feasibility.

3. The introduction of a prototype tool that helps automation of ScTDG.

4. The demonstration of the effectiveness of ScTDG compared to the current state-of-the-art automated test data generation approach, which provides evidence for its effectiveness compared to state-of-the-art.

5. The introduction of multi-objective formulation of service selection in ScTDG and its subsequent cost reduction, which includes the formulation for several QoS characteristics: cost, reliability, response time and availability.

6. The empirical evaluation of the multi-objective service selection in ScTDG using simulated scenarios with real-world parameters, which provides evidence for its ability to help reduce test data generation cost.

7. The introduction of cost-aware test suite minimisation in ScST and the empirical evaluation of the concept, which produces evidence for its applicability to real-world problems and to its ability to reduce runtime testing cost.

1.4 Technical and Scientific Challenges Faced in this Thesis

During the experimental evaluation of this thesis we faced several technical and scientific challenges. Some of these challenges were addressed and solutions to these challenges are presented in the following chapters. Some of the challenges could not be completely addressed during the duration of this thesis and left as future work. The challenges we faced can be grouped into three main groups:

1. Challenges posed by the lack of real-world case studies.

2. Challenges posed by the lack of established standards in SOA and service specifications.

3. Challenges posed by the lack of interoperability among tools, especially in the area of semantic services.

We faced with the challenges belong to the first group at every experimental analysis presented in the thesis. For example, while performing the experiments described in Chapter 4 not having semantically described services forced us to search for ontologies that describe the business domain of the services we used in our experiments. Unfortunately, most of the existing ontologies available on the internet are simple examples and do not adequately describe their domain. As a result, we created our own ontologies for the domain of the each service we used.

Not having real-world services with measured QoS metrics can also be included in this category. Experimental analysis presented in Chapter 5 and 6 are based on QoS measurements of services. However, we could not find adequate number of real-world services with semantic descriptions and measured QoS metrics. As a result, we generated synthetic QoS data (based on real-world examples) for our case studies.

The problem of non-established standards in SOA especially concerning semantic specifications is one of the major issues that prolong the transition to semantic services from traditional services [44]. We believe that this problem affects academic research as well. For example, as mentioned in Chapter 4 at the time of the experiments OWL-S did not fully support WADL specifications (for grounding). As a result, we were forced to convert all the RESTful services used in the experiments to SOAP services. Even though recent additions to OWL-S include support for WADL in OWL-S grounding, we are not sure if majority of the existing tools (parsers) are updated to include these new features.

In the subject of QoS metrics (for web services) not having established standards was also another challenge. As a result of not having established metrics and measurement definitions, we used the definitions from an earlier study [334]. The QoS characteristics in this study are defined using the most common terms in the existing work. Although, the definition and the used method of measurement for reliability metric that was provided in Chapter 5 is based on this study, it might not exactly agree with some of the existing work.

Unfortunately, best to our knowledge interoperability problems among semantic tools is a very common problem. During the experimental analysis of the solutions proposed in this thesis, we faced this problem several times. For example, the first time we faced this problem was during the selection of service broker. As we mentioned in Chapter 4, we selected Alive matchmaker [2] as the service broker. Even though, Alive is not the most widely used matchmaker according to the literature, it uses an a more recent OWL-S API which allows using more recent versions of OWL-S specifications. We could not use more popular matchmakers such as OWL-S/UDDI matchmaker and OWLS-MX due to their incompatibilities with the recent versions of OWL-S and limited OWL ontology support. One other related problem we faced was the limited XSLT transformation support in the earlier versions of OWL-S and parsers. The issues related to this problem forced us to use more recent version of OWL-S and tools that support it.

The most important challenge we faced in this thesis was the limitations posed by subsumption-based matchmaking. The challenges regarding this topic and our solutions to them are discussed in Chapter 4 in detail.

1.5 Assumptions Made in the Experimental Evaluation of the Proposed Solutions

We believe it is important to clearly state the assumptions we made throughout the experimental evaluation of the solutions presented in this thesis. As expected, we made several assumptions during the evaluation stage our solutions. In this section, we grouped the assumptions into three groups based on the experiment they relate to.

1.5.1 Automated Realistic Test Data Generation

In the experimental analysis of our solution to this problem (presented in Chapter 4), we made two assumptions:

1. **Existence of a semantic SOA where services are defined using OWL-S specifications and their business domain are described using OWL ontologies.** Unfortunately, non of the existing specifications are accepted as a standard to implement SOA. Our assumption in this matter is based on the popularity of the selected specifications in academic research and existence of tools that support mentioned specifications. Even though, in the future the specifications for SOA might be different than the specifications used in this experimental analysis of the solutions in this thesis, the general principal of SOA will remain the same. Thus, the solution presented in Chapter 4 will still be valid and applicable.

2. **Existence of services that provide data that can be used as test data.** In this thesis, we investigated the possibility of testing semantic applications with the data that can be acquired from existing services. The solution that we presented in Chapter 4 can only automatically generate test data in the case of availability of services that provide the required test data. This issue is identified as a limitation in the chapter. However, as we mentioned in Chapter 7, with concepts such as Data as a Service data is becoming easily accessible [91] and the growth in the investment in service and service related technologies increases the transition of businesses to services. In the light of these fact, we believe it is safe to assume that in the future more services providing data that can be used for testing will be available.

1.5.2 Cost Reduction in Test Data Generation

In the experimental analysis of our solution to this problem (presented in Chapter 5), we made three assumptions:

1. **Existence of a semantic SOA where QoS metrics are measured and available to the SOA stakeholders**. Even though, the literature suggest several models

and measurement techniques, best to the authors knowledge currently there is no established QoS standard for services. However, QoS is already defined and standardised in other fields such as telecommunication and networking. It is safe to assume that in the future a QoS standard for SOA will be established.

2. **For the models used in the experiments, we did not assume a perfect world**. As a result, each service population includes certain amount of noise. We believe, expecting a perfectly correlated population would be very unrealistic assumption and hinder our efforts to make our case studies as realistic as possible. In the experimental analysis, in order to sustain realism we created several levels of correlation in the populations and investigated the effects of the level of correlation in all possible combinations.

3. In the assignment of QoS metrics for the services in each population we two assumption. The first assumption we made were the **limits imposed on reliability values**. We defined reliability values to be between 0.50 and 0.99. The values selected conform with the reliability study we presented in Chapter 6, however the minimum reliability is specifically reduced to cover more possible scenarios. The second assumption we made is the **positive correlation between price and reliability**. As we mentioned in the chapter, it is hard to predict the cost and reliability relation. However, the general assumption in the engineering field that higher reliability is often comes with higher price and this is positive correlation is also perceived by the general public [333]. In the light of these information we followed the same trend and introduced a positive relation between price and reliability.

1.5.3 Cost Reduction in Runtime Testing

In the experimental analysis of our solution to this problem (presented in Chapter 6), we made three assumptions:

1. **Testing is performed by the integrator**. In SOA, it is likely that a service or

a composition is tested by multiple stakeholders. However, for service compositions structural testing or model-based testing can only be performed by the integrator who has access to the code. In this case, it is safe to assume that the solution presented in this chapter can only be used by an integrator.

2. We assume that the **stubs that can mimic the functionality of the invoked services can be generated**. This assumption is based on the discussion presented in Chapter 2 where we summarise the existing work on automatically generating functional service stubs.

3. **Selecting United States Postal Service ZIP Code validation service as ground truth in reliability analysis** presented in Chapter 6. We discussed the reasons for this assumption in detail in the related section in detail.

1.6 Overview of this Thesis

The thesis is organised as follows:

Chapter 2 - Literature Survey provides a survey of the existing work on the testing and verification of ScS. The chapter begins with the introduction of the concepts, technologies and perspectives in SOA. The rest of the chapter surveys testing methodologies such as unit testing, integration testing, regression testing, interoperability testing and testing techniques such as test case generation, fault-based testing, model-based testing and collaborative testing.

Chapter 3 - Analysis of Trends in Service-centric System Testing provides an analysis of existing and future trends in ScST area. The chapter first introduces an analysis of current trends. Then, the chapter provides an overview of the existing work, highlighting the area of testing, technologies that they are applicable to and case studies used in their experiments. Finally, the chapter provides an analysis of emerging trends in SOA and SCST.

Chapter 4 - Automated Realistic Test Input Generation introduces the concept of

ScTDG. The chapter first explains what 'realistic input' is and discuses the importance of it. Then the chapter explains ScTDG in detail, including service discovery, service grouping and composition building. Later, the chapter introduces the prototype tool that is used in the experiments and details how it enables functionalities such as service elimination and tester-specified test data generation. Finally, the chapter presents the empirical evaluation of ScTDG for two different realistic inputs using several real-world services.

Chapter 5 - Cost Reduction Through Multi-objective Data Source Selection introduces our multi-objective optimisation solution to the service selection problem in ScTDG. The chapter begins with a presentation of the subject background, including a summary of existing work in multi-objective test data generation, concept of QoS and multi-objective optimisation. Then the chapter introduces the details of multi-objective test data generation in ScTDG, including objective functions for QoS parameters, representation and selected algorithm. Later, the chapter presents the case studies used, the method of experimental validation and the research questions asked in the experimental validation. Finally, the chapter presents the empirical evaluation of the approach and the answers to the research questions.

Chapter 6 - Cost Reduction Through Pareto-optimal Test Suite Minimisation introduces the cost-aware multi-objective formulation of test suite minimisation for ScS. The chapter first discusses the need for test suite minimisation in ScST. Furthermore, it explains the concept of test suite minimisation, the multi-objective algorithm chosen and details of our approach. Then, it presents the case studies used, the method of experimental validation and the research questions asked in the experimental validation. Finally, the chapter presents the empirical evaluation of the approach and the answers to the research questions.

Chapter 7 - Conclusion completes this thesis with the summaries of its achievements and proposals of the possible future work.

Chapter 2

Literature Review

2.1 Background

This chapter introduces the concepts and technologies used in SOA and Web services. This chapter is aimed at software testing researchers who are unfamiliar with these concepts.

2.1.1 Definition of Service and Service-Oriented Computing

According to Papazoglou [249], SOC is a new computing paradigm that utilises services as lightweight constructs to support the development of rapid, low-cost and easy composition of distributed applications. This is a widely used definition of SOC.

The concept of a 'service' is comparatively elusive and consequently harder to describe. The definition adopted by the author is:

> *"Services are autonomous, platform-independent computational elements that can be described, published, discovered, orchestrated and programmed using standard protocols to build networks of collaborating applications distributed within and across organisational boundaries." [250].*

This definition describes services in terms of SOA, therefore it captures the services' technological aspects. In order to justify this description, the terms used need to be explained a little further.

According to the Oxford English Dictionary [243], the meaning of autonomy is:

autonomy

1. Of a state, institution, etc.: The right of self-government, of making its own laws and administering its own affairs. (Sometimes limited by the adjs. local, administrative, when the self-government is only partial; thus English boroughs have a local autonomy, the former British colonies had an administrative autonomy; political autonomy is national independence.)

   (a) Liberty to follow one's will, personal freedom.

   (b) Metaph. Freedom (of the will); the Kantian doctrine of the Will giving itself its own law, apart from any object willed; opposed to heteronomy.

2. Biol. Autonomous condition:

   (a) The condition of being controlled only by its own laws, and not subject to any higher one.

   (b) Organic independence.

3. A self-governing community (cf. a monarchy).

As this definition suggests, autonomy includes self-government, self-control, independence, self-containment, and freedom from external control and constraint. As a result, there are different definitions of autonomy in software. One of these definitions that suits the autonomy of services comes from Erl. According to Erl [100], autonomy, in relation to software, is a quality that "represents the independence with which a program can carry out its logic".

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

Other sources [21, 76, 197] also define autonomy of services in a manner similar to Erl. For example, Bechara [21] claims that services need to be autonomous in the sense that their operation is independent from other co-operating services. According to Bechara, this kind of autonomy is needed to enforce reusability of services.

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

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

The reason for having different definitions for services lies in the context in which services are used. For example, some researchers [70, 122, 190, 266] define services in the context of GRID Computing, and their definition focuses on the aspects of GRID technology. On the other hand, Jones [156] defines services in business terms and claims that defining a service from solely a technological point of view is insufficient. According to Jones, a service description must include other aspects that cannot be measured or defined purely by technology alone.

2.1.2 Characteristics of Service-Oriented Computing

The characteristics of SOC applications claim to deliver advantages over the traditional distributed applications. Such advantages include platform independence, autonomy and dynamic discovery and composition. There are two primary characteristics of SOC applications through which these advantages may arise [322]. These characteristics are:

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

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

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

2.1.3 Service-Oriented Architecture (SOA)

SOA is described as a strategy for building service-oriented applications. Its aim is to provide services that can be used by other services. In SOA, there are three main participants: a service provider, a service user and a service broker. These three participants perform the three fundamental SOA actions: publish, find and bind. Figure 2.1 illustrates this foundational SOA concept, its participants and the operations among participants.

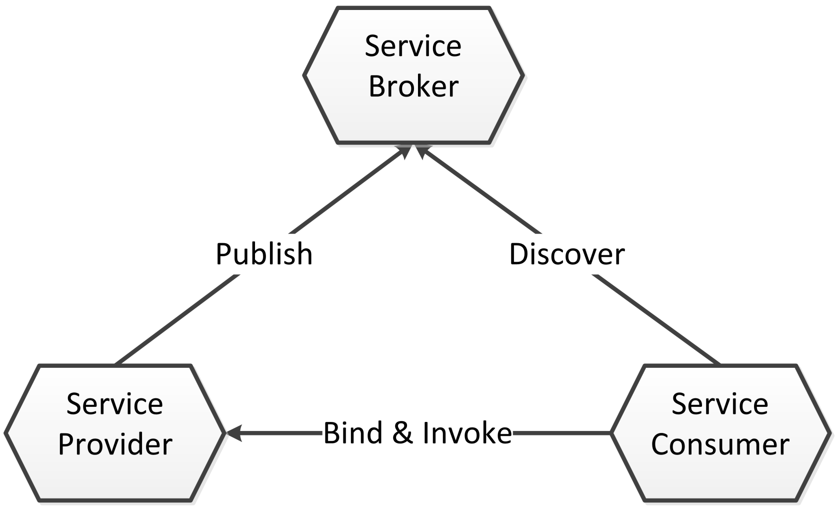


Figure 2.1: Service-Oriented Architecture

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

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

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

2.1.4 Web Services and Web Service Technologies

A web service (also referred to as service(s) hereafter) is defined as "a software system designed to support interoperable machine-to-machine interaction over a network" by W3C (World Wide Web Consortium) [350]. The aim of the Web services platform is to provide the required level of interoperability among different applications using pre-defined web standards. The Web services integration model is loosely coupled in order to enable the required flexibility in integration of heterogeneous systems.

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

The idea of SOA is older than that of Web services, but the 'great leap' of SOA has been facilitated with the introduction of Web services. Web services are generally accepted as the core element of SOA, providing the necessary autonomy, platform-independence and dynamic discovery and composition. Through web services, pre-existing systems can exchange information without the need to know any technical information about the exchange partner. Figure 2.2 describes the web service architecture and its core specifications that are used in performing each SOA operation. It also illustrates the way in which web services implement the general concept of SOA.

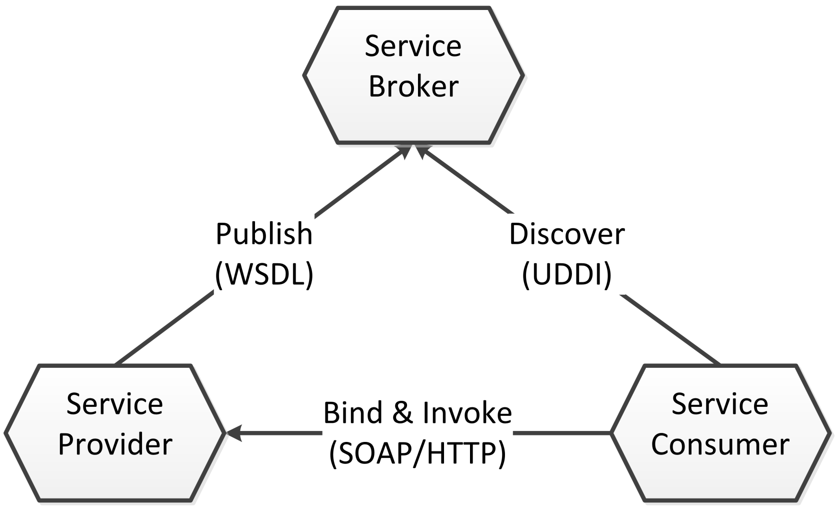


Figure 2.2: Web Service Architecture

In order to provide platform-independent messaging across a network, Web services architecture uses three core specifications:

1. *Simple Object Access Protocol (SOAP)*: SOAP is an XML-based protocol that allows data exchange over the Hypertext Transfer Protocol (HTTP). The W3C definition of SOAP is "a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment" [297]. Since SOAP as a protocol combines platform independent XML and HTTP, SOAP messages can be exchanged between applications regardless of their platform or programming language. The SOAP protocol allows exchange of XML messages with structured information between a web service and its users.

2. *Web Service Description Language (WSDL)*: W3C defines WSDL as "an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information" [349]. A WSDL specification is a web service interface that provides users with all the information they need, such as message formats, operations provided by

the web service and location of the web service, to interact with the service in the pre-defined standards.

3. *Universal Description Discovery and Integration (UDDI)*: UDDI is defined as "a set of services supporting the description and discovery of businesses, organizations, and other Web services providers, the web services they make available, and the technical interfaces which may be used to access those services" by OASIS (The Organization for the Advancement of Structured Information Standards) [329]. UDDI is an industry initiative that enables businesses to publish their services and enables potential customers to discover these services. UDDI registries can be either private or public. A web service is not required to be registered to any UDDI registry in order to be used.

There are also other Web services technologies that enable composition of multiple services, such as the Business Process Execution Language (BPEL) and Semantic Web Services (SWS), that work towards automated SOA integration.

BPEL is an executable language that defines and manages business processes that involve multiple web services [251]. BPEL is an XML-based flow language for composing multiple web services. Using BPEL, organisations can automate their business processes by orchestrating services within their network and the services from other businesses. The ability to use services from other organisations allows enterprises to build complex processes that include multiple organisations.

Unfortunately, traditional web services (WSDL services) provide only syntactic interfaces and current UDDI brokers support only index word-based searching of services with required attributes. Using traditional web services, automated discovery and invocation is almost impossible. In order to achieve automation, service descriptions must be machine readable. SWS have emerged aiming to solve the problems regarding automation. SWS are services with extended specifications that contain semantic information on input data and service behaviour. The aim of SWS is to provide a richer semantic specification for web services. At present there are several semantic web service proposals that include initiatives and projects such as WSDL-S [344], OWL-S [242], WSMO [343], METEOR-S [218] and SWSA/SWSL [285].

2.2 Perspectives in Service-centric System Testing

As expected from a distributed environment, SOA has multiple parties involved. Canfora and Di Penta [48] have specified five parties in SOA (the standard three SOA parties are depicted in Figure 2.1). These parties are the developer, the provider, the integrator, the third-party certifier and the end-user.

Developer: The developer is the party who implements the service and has the sole responsibility for the service evolution. There might also be situations where more parties are involved in service evolution, but generally the developer is accepted as the sole party to have access to the source code. Thus, developer side testing is typically performed in a white-box manner. In SOA, the developer is the only party who can perform structural testing on services. Being able to perform structural testing, a developer can perform thorough functional testing with minimal cost.

Provider: The provider is the party who deals with QoS in the SOA. The provider ensures that the service operates within the conditions defined by the SLA. The provider can perform structural tests if he has access to the source code. In many cases, the provider might not have the access to the source code (e.g. due to being in a different organisation from the developer) so he can only perform functional tests. In any scenario for both structural and functional tests the cost of testing is minimal for the provider. The provider can also perform non-functional tests. The main disadvantage of the provider could be not having a test suite with realistic inputs.

Integrator: The integrator is the party who uses existing services in his composition or application. The integrator might not have access to the source code for all the services in the system. As a result, the integrator has to test the services to verify their behaviour with the possible inputs and scenarios of the composition (black-box testing). The integrator performs tests to observe both the functional and non-functional performance of the services involved and the service-centric system. The integrator can perform very realistic non-functional tests as opposed to the provider and the developer. The advantages of SOA, such as automated service discovery and binding, bring testing challenges to the integrator. The integrators not having control over the evolution of the service can also add to this challenge. The cost of testing for the integrator can be high due to the number of invocations required to test his system.

Third-party Certifier: The third-party certifier is the party who provides testing services to the other parties. Such third-party testing increases trust in the service and can be more assuring to the integrator rather than testing performed by the provider or the developer. Even when considering any added testing fees involved, third-party testing might reduce the overall testing costs for the provider. Third-party certification might be an efficient solution in SOA to reduce the amount of testing performed on each service. However, third-party testing might not be as effective as integrator testing since third-party testing is performed with a possibly different composition to that of the integrator.

End-User: The end-user is the party who uses the service through an application or platform. The end-user is not directly involved in testing but the effects of testing can affect his user experience.

2.3 Test Case Generation Approaches

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

2.3.1 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 web 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 datatype information.

Many proposed approaches [16, 20, 29, 130, 185, 201, 235, 295] for WSDL-based test case generation are based on the XML Schema information. Test cases for each service operation are generated based on 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 datatype information, with various constraints for each input type, allows generation of test data for each simple type. In XML, complex datatypes can also be defined. Test data generation for complex datatypes simply requires decomposition of the complex type into simple types. Test data is generated for each of these simple types and the combined data is used as complex test data.

Li et al. [185] propose a test case generation method that combines information from WSDL specifications and user knowledge. They introduce a tool called WSTD-Gen that supports this method. The tool allows users to customize data types and select test generation rules for each datatype.

Chakrabarti and Kumar [65] propose an approach that aims to generate test cases for testing RESTful web services. The authors also introduce a prototype tool that supports this approach.

Proposed approaches [201, 235, 295] 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 web services. However, there are cases that might require test cases that run multiple methods. An example of this situation is an operation that requires the execution of another operation, such as login, as a precondition. Bai et al. [16] address this problem by using data dependencies among the provided operations. The mapping of dependencies is based on the input and output messages of different methods.

Test data generation using WSDL definitions is limited to input datatypes due to 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, since they contain more information compared to WSDL. The use of semantic model OWL-S for test data generation is proposed [17, 77, 326, 338] not only because of the behavioural information it contains, but also because of the semantic information concerning the datatypes on which the service operates. This semantic information, in the form of ontology allows, ontology-based test data generation [338].

Ye et al. [369] introduce a static BPEL defect analysis system focusing 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 [42]. Bozkurt and Harman [42] categorise the test data required by online systems as ‘Realistic Test Data’ (RTD) and define it as *data that is both structurally and semantically valid*. The authors discuss the importance of using RTD in ScST and claim 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 [74]. Their approach generates test data using applications with Graphical User Interfaces (GUI). This approach harnesses user input data from GUI elements and uses the harnessed data to generate test cases for ScS.

An automated solution is proposed by Bozkurt and Harman [41, 42]. The proposed approach is capable of generating RTD while providing a high level of automation. The authors also present a framework that supports the approach called ATAM service-oriented test data generator. The approach exploits existing web services as sources of realistic test data 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 based on user-specified constraints. Bozkurt and Harman [43] also proposed the use of multi-objective optimisation and QoS parameters within their approach in order to reduce the cost of test data generation and testing, as well as increasing the reliability of the testing process.

2.3.2 Contract-based Test Case Generation Approaches

Design by Contract (DbC) [219] is a software development approach, where contracts define the conditions (pre-conditions) for a component to be accessed. Contracts also include conditions (post-conditions) that need to be held after the execution of methods of that component with the specified pre-conditions. Using contracts, some unexpected behaviour of the SUT can be detected and the information from contracts can also be used to enhance the testing process itself. Software testing using contracts has been applied to traditional software by many researchers [153, 184, 187, 228].

Since 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 pre and post-conditions of operations, increases the testability of services.

Heckel and Lochmann [136] propose the use of the DbC approach for Web services and discuss the reasons for contracts being implemented at different levels, such as implementation-level, XML-level and model-level. The contracts defined at model-level are derived from model-level specifications and the reason for this is to minimize the effort required to generate contracts. The created contracts are later used in unit testing of services to check if the service conforms to its specifications. Using contracts, the proposed testing approach enables automated creation of test cases and test oracles.

Atkinson et al. [14, 15] propose a technique called test sheets in order to generate unit test cases and test oracles. Test sheets contain contract information which identifies the relation between the operations of a service. The included relations define the effects of each operation from the clients perspective in order to help validation.

WSDL extensions proposed to allow WSDL files to accommodate contract information such as pre- and post-conditions. For example, Tsai et al. [324] discuss the reasons for an extension to WSDL in order to perform black-box testing, and propose an extended WSDL that carries additional information such as input-output dependency, invocation sequence, hierarchical functional description and sequence specifications. Similarly, Mei and Zhang [212] propose an extended WSDL that includes contract information for the service and also a framework that uses this extended WSDL to test services. Heckel and Lochmann's [136] XML-level contracts also require an extension to WSDL.

Noikajana and Suwannasart [232] propose the use of a Pair-Wise Testing (PWT) technique to facilitate contract-based test case generation. In the proposed approach, pre- and post-conditions are included in Web Service Semantics (WSDL-S) specifications using the Object Constraint Language (OCL).

Askarunisa et al. [11] propose the use of PWT and Orthogonal Array Testing (OAT) in test case generation for semantic services. The authors also compare these two approaches in order to determine which approach performs better in different testing scenarios. The authors use the same contract specifications (OCL and WSDL-S) as Noikajana and Suwannasart.

Liu et al. [193] also propose 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. [205] propose the inclusion of contract information in service stubs. These semantically extended stubs carry contract-like information such as pre- and post-conditions.

Dai et al. [77] propose contracts that can be contained in OWL-S process models. Proposed contracts carry information such as pre- and post-conditions between a service user and a service. Dai et al. also present a framework that is capable of generating test cases and oracles, monitoring test execution and verifying the SUT. Bai et al. [17] propose a testing ontology model that describes test relations, concepts and semantics, which can serve as a contract among test participants.

Saleh et al. [278] propose 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.

2.3.3 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 [353]. In mutation testing, faults are introduced into every subdomain that is known to function correctly, in order to measure the effectiveness of the test suite. By introducing a fault into a subdomain, it is possible to identify the test cases belong to that subdomain during test executions.

Heckel and Mariani [137] claim that Δ-Grammars are more suitable for testing web services than UML diagrams, due to their ability to describe the evolution of services. Heckel and Mariani suggest a partition-testing approach based on WSDL definitions and Δ-Grammars. Similarly, Park et al. [253] also apply this approach to service selection.

The application of partition testing to web services is proposed at two different levels. Bertolino et al. [29] propose the use of the category-partition method [353] with XML Schemas in order to perform XML-based partition testing. This approach automates the generation of test data using XML Schemas. Bertolino et al. introduce a tool that supports this approach called TAXI. Another approach is proposed by Bai et al. [17] for OWL-S semantic services. Bai et al. introduce 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. Sun et al. [305] propose an approach that combines random testing and partition testing. The authors introduce a framework that automates the generation of partitions and test cases for each partition.

2.3.4 Experimental Results

This section also follows the classification from the previous section.

2.3.4.1 Experimental Results of Specification-based Approaches

Bartolini et al. [20] 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, managing to kill 63% more mutants than a manually generated test suite using an enterprise testing software.

Ma et al.'s [201] constraint-based test data generation approach is validated using three versions of a synthetic web service. In this context, synthetic web service means a web service implemented to test the testing approach. During the experiments, 4,096 different test data is generated for a complex datatype and revealed two bugs in two different services.

Bai et al. [16] experimented on 356 real web services with 2050 operations in total. For the 2,050 operations, they generated 8,200 test cases to cover all operations and valid messages. Test cases were generated to perform two analyses (constraint and boundary analysis) on datatypes and operation dependency analysis. According to results, 7,500 of the generated test cases were exercised successfully on 1,800 operations. The authors' explanation for the unsuccessful test cases are the mistakes in WSDL documents.

Conroy et al. [74] 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 to their approach. According to the authors, it took two hours to write a parser and ten minutes for their approach to generate test cases.

Bozkurt and Harman [42] experimented on 17 existing commercial web services in two different case studies. Testing is performed on 4 of these services while 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 an 8% success rate in case study 1, a 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 based on tester constraints and the approach achieved 100% success rate for all of the given constraints in both of the case studies.

2.3.4.2 Experimental Results of Contract-based Approaches

Mei and Zhang [212] experimented on two synthetic web services: triType and mMiddle. The test suite created by this approach achieved 95% mutation score and 81.5% to 96% reduction in test cases while maintaining the test coverage.

Askarunisa et al. [11] experimented on 2 real and 2 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 to random test data generation. Both PWT and OAT have shown significant ability to reduce the number of required test cases (up to 99.76% reduction using PWT and 99.84% reduction using OAT).

Mani et al. [205] experimented on a set of services from an IBM project and their approach achieved 37% reduction in test suite size.

Noikajana and Suwannasart [232] experimented on two example web 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 suite generated by this approach achieved 63%

(RectangleType) and 100% (IncreaseDate) mutation score, whereas test suite 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).

2.3.4.3 Experimental Results of Partition Testing Approaches

According to Bai et al.'s experiments, using partitioning, a 76% reduction is achieved; reducing 1,413 randomly generated tests for 17 partitions to 344. Comparing Bai et al.'s partition technique against random generation also shows the effectiveness of this approach. In order to cover the 1,550 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.

2.3.5 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 invalid test data with the aim of measuring the robustness of services. The approaches aiming to test the robustness services are discussed in detail in Section 2.5.

Specification based test data generation approaches focus on generating test data to perform boundary analysis or constraint-based tests. These test 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 above is the type of test data they are able to generate. Almost all the approaches use XML datatype constraints and other constraints provided by either the tester or the provider and generate test cases using them. However, none of them aim to generate realistic test data except Conroy et al. [74] and Bozkurt and Harman [42].

Unfortunately, running a large number of test cases might be a problem in WST due to the cost of invoking services or access limitations. For example, to the stakeholders for whom testing cost is minimal, such as the developer and the provider, 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 approaches from Section 2.3.1. Partition testing can be very useful to the integrator and the certifier for whom testing cost is high.

Contract-based WST can achieve better and more efficient testing than WSDL-based testing due to 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 [77, 136, 212, 324] propose extensions to standards like WSDL and OWL-S in order to solve this problem. Many existing SWS proposals already include contract like information, such as pre- and post-conditions and their effects on execution. However, further additions to SWS may be required in order to improve testability.

2.4 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 to be units to be tested. A service composition or choreography may also considered as a service unit.

Service-Centric Unit Testing (SCUT) is generally performed by sending and receiving SOAP or HTTP messages. The tester generates the SOAP/HTTP messages for the operation/application under test 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 [299], SOAP Sonar [296], HP service Test [140] and the Oracle Application Testing Suite [238]. Even though these tools help to 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 have 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.

2.4.1 Perspectives in Unit Testing

SCUT 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 web services. The integrator may also perform simulations by using stub or mock services to test the business process.

2.4.2 Unit Testing Approaches

The need for tools that can automate unit testing has been addressed by the research community. For example, Sneed and Huang [295] introduce a tool called WSDLTest for automated unit testing. WSDLTest is capable of generating random requests from WSDL schemata. WSDLTest is also capable of verifying the results of test cases. This capability is achieved by inserting pre-conditions and assertions in test scripts that are manually generated by the tester. The provided verification method requires the tester to be familiar with the SUT in order to generate the necessary assertions.

Lenz et al. [176] propose 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.

Zhang et al. [386] present a framework based on Haskell modules that is capable of generating and executing test cases from WSDL specifications. The HUnit component organises the test plans and provides unit test execution.

One of the main problems of software testing is the oracle problem [290, 320]. 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 is addressed by Chan et al. [66]. Chan et al. propose a metamorphic testing framework that is capable of performing unit testing. Metamorphic testing [72] 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 propose the use of metamorphic services that encapsulates a service and imitates its functionality. Verification for the SUT is provided by the encapsulating metamorphic service that verifies the input and output messages against the metamorphic relations. Chen et al.'s framework enables web service users to determine the expected results, but requires the use of relations which can be costly for the provider. Sun et al. [306] also introduced another automated framework for metamorphic testing of services.

Similarly, Heckel and Lochmann [136] propose the generation of test oracles using pre-generated contracts. The contracts, created using the DbC approach, are supplied by the provider and carry information such as pre- and post-conditions.

Atkinson et al. [15] propose the use of a technique called *test sheets* in order to generate unit test cases and test oracles. The test sheets approach uses tables that define test cases similar to the Framework for Integrated Test (FIT) [108], a framework for writing acceptance tests. Two types of test sheets are used in this approach: an input test sheet that contains specifications defining a set of test cases and a result test sheet that contains outputs from SUT for the test cases in the test sheets. Atkinson et al. also include contract information that identifies the relation between the operations of a service, defining their effects from the clients' perspective in order to help validation.

An automated solution to the oracle problem is proposed by Tsai et al. [319]. Tsai et al. propose the adaptation of blood group testing to web services and call this technique Adaptive Service Testing and Ranking with Automated oracle generation and test case Ranking (ASTRAR) [317, 321]. ASTRAR is similar to n-version testing, where multiple web services that have the same business logic, internal states and input data are tested together with the same test suite. Even though the main goal of group testing is to test multiple web services at one time (to reduce the cost of testing and increase the efficiency), it also helps in solving the reliable test oracle problem within its testing process. Tsai et al. [328] also proposed an extension to group testing that allow services to be tested in parallel. The proposed approach uses cloud and a service-level MapReduce technique.

Yue et al. [380, 381] propose a message-based debugging model for web services. The authors present 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.

Zhu et al. [398] introduce a testing framework called SCENETester focusing on fault localisation for service compositions. The authors propose an execution flow model for compositions that helps identifying the source of the fault. The framework automates service validation using a virtual runtime environment and performs concurrent testing using distributed nodes.

Unit testing of web service compositions using BPEL has also been addressed in the literature. According to Mayer and Lübke [208], 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, web service stubs or mocks can be used instead of the web services that participate in the process. Mayer and Lübke [208] propose a framework that is capable of performing real-world unit testing. This framework can replace participating web services with service mocks. The framework also provides a mechanism for asynchronous messaging by providing an implementation of WS-Addressing [345]. Li et al. [186] adopt a different approach for unit testing in which BPEL processes are represented as a composition model. Similar to Mayer and Lübke's framework, Li et al.'s framework uses stub processes to simulate the parts that are under development or inaccessible during the testing process.

Mani et al. [205] propose the idea of semantic stubs. Semantic stubs carry additional information such as pre- and post-conditions. Semantic stubs enable input message verification, expected output generation and exception message generation for the simulated services. The authors also present a framework that automatically generates stubs from semantically annotated WSDL descriptions.

Palomo-Duarte et al. [247] introduce 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. [146, 145] introduce 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 [270] introduce a framework that is aimed at simulating RESTful web services. Li et al. [182] introduce a toolkit called SOArMetrics for evaluating SOA middleware performance and application testing. Chandramohan et al. [67] introduce a tool that simulates distributes web server environments called Distributed Web Service Evaluator (DWSE). DWSE helps evaluating service behaviour in distributed server environments with the help of the included test kit. Hallé [127, 128] propose an approach to simulate SOAP services using temporal logic specifications. The author introduce an extended linear temporal logic called LTL-FO$^+$ which is used in describing input/output parameters, messages and ordering constraints between messages. The approach generates an on-the-fly service response of the simulated service using previous interactions and a custom symbolic satisfiability algorithm.

2.4.3 Experimental Results

Sneed and Huang [295] experimented on an eGovernment project with nine web services. They generated 22 requests per service and 19 out of 47 verified responses contained errors. They revealed 450 total errors in the whole project, of which 25 of them were caused by the services.

Tsai et al. [317] experimented on 60 different versions of a synthetic real-time web 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 claim 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. [205] 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 to remote services.

2.4.4 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, though they lack automated test oracle generation.

Tsai et al. [319], Chen et al. [394] and Heckel and Lochmann [136] address the oracle problem. Tsai et al.'s approach 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 Mayer and Lübke [208], Mani et al. [205], Van Gilst [270], Ilieva et al. [146] and Li et al. [186] 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.

2.5 Fault-Based Testing of Service-centric Systems

According to Morell [223], 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 non-existence of known faults rather than trying to find unknown faults that do exist.

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 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 [130] classified test data generation for different testing techniques and also surveyed fault-based testing research in the web services domain. These testing techniques are:

1. *Interface propagation analysis* that is performed by randomly perturbing the input to a software component.

2. *Boundary value based robustness testing* where test data is chosen around the boundaries of the input parameter.

3. *Syntax testing* with invalid input where the rules of the specification of the input parameter are violated.

4. *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 section, 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.

2.5.1 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. On the other hand, 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. However, 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 since they require generation of many mutants and running each mutant with many test cases to kill it.

2.5.2 Fault-Based Testing Approaches

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

2.5.2.1 XML/SOAP 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 simply by sending a faulty SOAP message to the web service. After perturbations, the web service's behaviour with the faulty message is observed for verification.

One of the earliest examples of SOAP perturbation is proposed by Offutt and Xu [235]. Offutt and Xu propose three different types of perturbations:

1. *Data Value Perturbation* (DVP) that is performed by modifying the values in a SOAP message.

2. *Remote Procedure Calls Communication Perturbations* (RCP) that is performed by modifying the arguments of the remote procedures. Offutt and Xu propose the application of mutation analysis to syntactic objects and data perturbation to SQL code. SQL code perturbation also facilitates SQL injection testing.

3. *Data Communication Perturbations* (DCP) that is used for testing messages that include database relationships and constraints

Xu et al. [364] propose an approach where perturbation is applied to XML Schemas in order to generate test cases. Xu et al. define XML Schema perturbation operators for creating invalid XML messages by inserting or deleting data fields. Almedia and Vergilio [79] also adopt the same approach and propose a tool called SMAT-WS that automates the testing process. Almedia and Vergilio also introduce some new perturbation operators for XML perturbation. Hanna and Munro [130] test the robustness of services by violating the input parameter specifications from WSDL files. Hanna and Munro's approach can test both the web service itself and the platform the service resides in. Zhang and Zhang [384] propose boundary value fault-injection testing in order to help select reliable web services. Similarly, Vieira et al. [332] propose a framework that applies fault-injection to the captured SOAP messages. Martin et al. [207] propose a framework called WebSob that tests web services for robustness. WebSob tests web service methods with extreme or special parameters. Li et al. [181] propose an approach that combines SOAP data perturbation and Combinatorial Test Data Generation (CTDG). The authors claim that the addition of CTDG to data perturbation increases the effectiveness of the testing which leads to a higher fault detection rate compared to data perturbation only method.

Salva and Rabhi [280] propose 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 thus it does not test the web service itself.

Tsai et al. address the problems of test data ranking and fault-based testing within a single approach. Tsai et al. [326] propose an approach based on boolean expression analysis that can generate both true and false test cases. The proposed approach is supported by a framework that can rank test cases according to the test cases' likelihood to reveal errors.

Wang et al. [339] propose 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.

Shafin et al. [287] proposed a fault-based testing method for OWL-S semantic services. The proposed approach generate abstract test cases from OWL-S specifications. The authors propose an extended OWL-S model which contains extra constraints in the form of SWRL expressions. These constraints are used in generating test data. These test cases then mutated using the eight mutation operators introduced by the authors.

2.5.2.2 Network Level Fault Injection

Network Level Fault Injection (NLFI) is a fault-injection approach in which faults are injected by corrupting, dropping and reordering the network packages. Looker et al. [196] propose the use of this technique along with a framework called the Web Service Fault Injection Tool (WS-FIT). At the network level, latency injection can be performed along with SOAP perturbation. WS-FIT can perform both SOAP perturbations and latency injections. Looker et al. also propose another fault-injection approach that simulates a faulty service. Faulty service injection is performed by replacing the values in SOAP messages with incorrect values that are within the specified range of the parameter. Looker et al. [195] also propose an extended fault model ontology that is used for generating faults and a failure modes ontology identifies the type of faults (seeded or natural fault).

Farj et al. [104] introduce a framework called Network Fault Injector Service (NetFIS). NetFIS is capable of generating both network fault injection and SOAP message perturbation.

Juszczyk and Dustdar [158, 159] introduce a SOA testbed based on the Genesis2 framework for testing ScS. The testbed is capable of simulating QoS of the participating web services and also generating issues such as packet loss and delay and service availability. The authors [157] also propose the use of aspect oriented programming to automate generation of replicas of services and redirect service invocations at runtime. The proposed extension is only applicable to Java-based systems.

2.5.2.3 Mutation of Web Service Specifications

One of the first examples of Web Service Mutation Testing (WSMT) was applied to WSDL specifications. Siblini and Mansour [291] propose the use of WSDL mutation for detecting interface errors in web services.

Mei and Zhang [212] define mutation operators for contracts. The contracts in this approach are included in the extended WSDL specifications that are 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 to WSDL mutation. For example, Lee et al. [174] propose an ontology-based mutation to measure semantic test adequacy for composite web services and for semantic fault detection.

Similarly, Wang and Huang [336, 337] propose another ontology-based mutation testing approach that uses OWL-S requirement model. Wang and Huang suggest a modified version of the requirement model enforced with Semantic Web Rule Language (SWRL) [309] 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 [7] and Watkins [340] propose 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. [112] 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 runtime.

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. [169, 171] present a public web service robustness assessment tool called wsrbench [359]. Wsrbench provides an interface for sending SOAP messages with invalid web service call parameters. Wsrbench is also capable of providing the tester with detailed test results for a web service. Laranjeiro et al. [170] also propose the use of text classification algorithms to automate the classification of the robustness test results. Carrozza et al. [56] claim that the existing robustness testing tools do not consider the testing needs of complex systems. The authors introduce a robustness tool called WSRTesting tool which aimed at applying robustness testing to complex systems.

Bessayah et al. [31] present 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. [86] introduce an automated mutant generator for BPEL processes called GAmera. GAmera uses a genetic algorithm to minimise the number of mutants without losing relevant information. GAmera is also able to detect potentially equivalent mutants. Boonyakulsrirung and Suwannasart [40] also introduced another framework aimed at generating weak mutants of BPEL processes. The proposed approach includes four different weak mutation operators for BPEL expressions, statements and blocks (blocks have two different type based on the number of expected iterations).

Oliveira et al. [237] claim that correctness of manual classification of robustness test results relies on expert knowledge and it might increase the total testing time. In order to address these issues, the authors propose an approach that automatically classifies the results of robustness tests. The approach aims to identify test cases that produced unexpected results. The approach classifies the responses into two categories: correct or crash. The classification is automated using rule-based classification and machine learning algorithms.

2.5.3 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 are inserted into the Mars Robot Communication System (MRCS) and 100 DVP, 15 RCP and 27 DCP tests are generated. The generated tests achieved 78% fault detection rate in seeded faults (14 out of 18) and also revealed two natural faults. Xu et al. also experimented on MRCS and additionally on the supply chain management application from WS-I, achieving 33% fault detection. Almedia and Vergilio ran their experiments on a system consisting of 9 web services and revealed 49 faults, of which 18 of them were seeded by SMAT-WS. Vieira et al. experimented on 21 public web services and observed a large number of failures. However, 7 of these services showed no robustness problems. Vieira et al. also highlight that a significant number of the revealed errors are related to database accesses. Tsai et al. experimented on 60 BBS web services with 32 test cases and in these experiments negative test cases revealed more faults than positive test cases. Salva and Rabhi [280] experimented on the Amazon E-Commerce service. 30% of the test cases generated using their approach caused unexpected results.

Looker et al. [196] experimented on a simulated stock market trading system that contains three web services. Baseline tests showed that latency injection caused the system to produce unexpected results 63% of the time. Faulty service injection results showed that the users do not encounter faults by the application of this injection method.

Domínguez-Jiménez et al. [86] experimented on the loan approval example. The authors tested GAmera with different configurations and mutant sizes ranging from 50% to 90% of the possible 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 2 equivalent mutants. In each population, between 2 and 3 of the generated mutants were not killed by the test suite.

Bessayah et al. [31] experimented on the Travel Reservation Service (TRS is an example BPEL process) composed of three web 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. [170] 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 SVM. 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 [212] WSDL mutation achieved 95%, Wang and Huang's [336] OWL-S mutation achieved 98.7% and Lee et al.'s [174] OWL-S mutation achieved 99.4% mutation score. Results also proved the effectiveness of mutation testing in test case selection. Mei and Zheng's approach achieved 96% and 81.5% reduction in test cases while maintaining the test coverage. Approach of Lee et al. also helped with equivalent mutant detection by detecting 25 equal mutants out of 686 generated mutants.

Since the aim of fault-based testing is to observe the behaviour of a system with faulty data, using fault-based testing error handling code can also be tested and verified. The information on the behaviour of a service under unexpected situation is valuable to the integrator to implement a robust service composition.

Fu et al.'s 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. Fu et al.'s approach can only be applied to Java web services and performed by the developer. This approach does not guarantee the execution of the error recovery code in the case of an error, neither the correctness of the recovery action.

2.6 Model-Based Testing and Verification of Service-centric Systems

Model-Based Testing (MBT) is a testing technique where test cases are generated using a model that describes the behaviour of the SUT. Advantages of model-based testing, such as automating the test case generation process and the ability to analyse the quality of product statically, makes it a popular testing technique. The formal and precise nature of modelling also allows activities such as program proof, precondition analysis, model checking, and other forms of formal verification that increase the level of confidence in software [93].

Formal verification of web service compositions is popular due to formal verification methods' ability to investigate behavioural properties. The earliest work on formal verification of web service compositions dates back to 2002. The existing work that has been undertaken in formal verification of web service compositions is compared by Yang et al. [368]. Morimoto [224] surveyed the work undertaken in formal verification of BPEL processes and categorized proposed approaches according to the formal model used. This chapter categorizes the work undertaken after 2004 until 2009 in MBT (including formal verification) of ScS according to the testing technique used.

2.6.1 Perspectives in Model-Based Testing

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

2.6.2 Model-Based Testing approaches

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

2.6.2.1 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, 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 (using constraint solving) (PA) are the earliest proposed MBT methods for ScST. The generation of Control Flow Graph(s) (CFG) from BPEL processes is widely adopted [98, 365, 379]. 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, while 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. [365] propose an automated test data generation framework that uses an extended CFG called Extended Control Flow Graph (XCFG) to represent BPEL processes. XCFG edges contain BPEL activities and also maintain the execution of activities. Similarly, Yuan et al. [379] propose a graph based test data generation by defining another extended CFG called BPEL Flow Graph (BFG). The BFG contains both the structural information (control and data flow) of a BPEL process that is used for test data generation and semantic information such as dead paths.

Lallai et al. [167, 168] propose the use of an automata called Web Service Time Extended Finite State Machine (WS-TEFSM) and Intermediate Format (IF) in order to generate timed test cases that aim to exercise the time constraints in web service compositions. An IF model, which enables modelling of time constraints in BPEL, is an instantiation of the WS-TEFSM. For IF model transformation from BPEL, Lallai et al. use a tool called BPEL2IF, and for test case 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 faults handlers and termination of BPEL process. Similarly, Cao et al. [51, 52] propose the use of TEFSM for BPEL processes. The authors also introduce a tool that automates the proposed approach called WSOTF.

Maâlej et al. [203, 204] proposed a flow graph based model called Timed Automata which include several constraints functional and temporal constraints for BPEL processes. The authors also introduce a prototype tool called WS-BPEL Compositions Conformance Testing (WSCCT) that automates the test data test data generation and execution. WSCCT is capable of simulating partner processes that are not available during testing.

Endo et al. [98] propose the use of the CFG approach in order to provide a coverage measure for the existing test sets. Endo et al. propose a new graph called the Parallel Control Flow Graph (PCFG) that contains multiple CFG representing a service composition and communications among these CFG. They also present a tool that supports the proposed technique called ValiBPEL.

Liu et al. [189] propose a flow-graph based model for OWL-S web services called OCFG which can be used with graph traversal techniques to generate test cases. OCFG is capable of modelling both atomic and composite processes and all OWL-S control constructs as well as preconditions and results.

Li and Chou [179] propose the application of combinatorial testing to stateful services. The authors propose a combinatorial approach that generates multi-session 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 [178] also propose an abstract guarded FSM to address testability issues in conformance testing.

Belli and Linschulte [23] propose a model-based testing approach for testing stateful services. The authors introduce a model that captures events with the corresponding request and response for each event called Event Sequence Graph (ESG). In the proposed approach, ESG are supported by contract information in the form of decision tables. The authors also introduce a tool that generates test cases from decision tables called ETES. Endo et al. [97] also propose the use of ESG in order to perform a grey-box testing, focusing on code coverage. Belli et al. [22] also extended ESG to ESG4WSC in order to test service compositions and introduced a new algorithm that generates test cases from ESG4WSC model. Wu and Lee [360] propose the use of ESGs to generate test cases SaaS services. In the proposed approach, source code comments are used to generate WSDL-S specifications. Then, WSDL-S documents are used in the generation of ESGs.

Hou et al. [138] address the issues of message-sequence generation for BPEL compositions. In this approach, BPEL processes are modelled as a Message Sequence Graph (MSG) and test cases are generated using this MSG.

Paradkar et al. [252] propose a model-based test case generation for SWS. In this approach test cases are generated using pre-defined fault-models and input, output, pre-conditions and effects information from semantic specification.

Guangquan et al. [125] propose the use of UML 2.0 Activity Diagram [258], a model used for modelling workflows in systems, to model BPEL processes. After modelling the BPEL process, a depth first search method combined with the test coverage criteria is performed on the model in order to generate test cases.

Ma et al. [202] propose the application of Stream X-machine based testing techniques to BPEL. A stream X-machine [172] is an extended 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.

Endo and Simao [99] propose the use of state models in order to test service compositions. In the approach the the state model is generated manually and there are other artefacts that are generated from service and composition specifications. The generated artefacts are used in automation of test case generation and execution.

Sun et al. [304] introduce another graph model for modelling BPEL compositions called BPEL Graph Model (BGM). The authors also propose an approach that generates test cases from BGM. Similar to the previous approaches test cases are generated using traversal algorithms based on the testers coverage criteria.

Casado et al. [57, 58, 59, 61] claim that there is no practical approach to test long-lived web service transactions and propose an approach that aims to generate test cases for testing web service transactions. The authors introduce a notation and system properties for testing the behaviour of web service transactions that comply with WC-COOR and WS-BA. In this approach, test cases are generated using Fault Tree diagrams. The authors [60] also proposed a framework that automates the testing of WS transactions.

Search-based test data generation has attracted a lot of recent attention [1, 131, 209]. Search-based test data generation techniques enable automation of the test generation process, thus reducing the cost of testing. Blanco et al. [37] propose 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. [310, 312] propose two new models for describing composite web 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 web services. Tarhini et al. propose three different sets of test case generation for testing different levels of web service composition. Test cases in the first set aim to perform boundary value testing using the specifications derived from the

WSDL file. The second set is used for testing the behaviour of a selected web service. The third set tests the interaction between all the participating services and test cases for this set are generated using TPG and TLTS.

2.6.2.2 Model-Based Testing & 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 [165]. In symbolic testing, the SUT is executed symbolically using a set of classes of inputs instead of a set of test inputs. A class of inputs, represented as a symbol, represents a set of possible input values. The output of a symbolic execution is generated in the form of a function of the input symbols. An example method that generates test cases using symbolic execution is the BZ-Testing-Tools (BZ-TT) method [175]. The BZ-TT method takes B, Z and statechart specifications as inputs and performs several testing strategies on them, 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 Web services using symbolic execution is proposed by Sinha and Paradkar [292]. Sinha and Paradkar propose an EFSM based approach to test the functional conformance of services that operate on persistent data. In this approach, EFSMs are generated by transforming a WSDL-S model into an EFSM representation. Sinha and Paradkar propose the use of four different test data generation techniques: full predicate coverage, mutation-based, projection coverage and the BZ-TT method. For full predicate coverage, each condition of the EFSM is transformed into disjunctive normal form [85] 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. [26] propose another approach that uses symbolic execution in order to test web service compositions. In this approach, a web service composition is first translated into a STS, then a Symbolic Execution Tree (SET) is created using STS of the composition. Bentakouk et al.'s approach takes coverage criteria from the tester and generates the set of execution paths on SET. These generated paths are executed using a test oracle over the service composition. Bentakouk et al. claim that using symbolic execution helps avoid state-explosion, over-approximation and unimplementable test case problems that are caused by labelled transition systems. As a result, Bentakouk et al.'s approach can handle rich XML-based datatypes.

Zhou et al. [393] propose an approach aimed to test service compositions using Web Service Choreography Description Language (WS-CDL). The approach uses a Dynamic Symbolic Execution (DSE) method to generate test inputs. In the approach, assertions are used as test oracles. The approach uses an SMT solver for generating inputs that satisfy path conditions.

Escobedo et al. [101] propose an approach for testing BPEL processes using a specialised 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.

2.6.2.3 Model-Based Testing & 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. [73]. 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 [55] or Finite State Machines (FSMs) [119] , 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 SPIN [300], NuSMV [233] or BLAST [38].

Fu et al. [114] propose a method that uses the SPIN model-checking tool. The proposed framework translates BPEL into an XPath-based guarded automata model (guards represented as XPath expressions [362]) 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 [262] with bounded queues (directly), or with synchronous communication based on the result of the synchronizability analysis. The SPIN tool takes Promela as the input language and verifies its Linear Temporal Logic (LTL) [272] properties. Interactions of the peers (participating individual web 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. [117] use a similar method to that of Fu et al. [114] in order to generate test cases. García-Fanjul et al.'s approach differs from Fu et al.'s approach in transforming BPEL directly to Promela. García-Fanjul et al. generate test cases using the test case specifications created from counterexamples which are obtained from model-checking. LTL properties are generated for these test case specifications, aiming to cover the transitions identified in the input. Transition coverage for a test 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. [392] propose another test case generation method using model-checking for web service compositions. Test coverage criteria such as state coverage, transition coverage and all-du-path coverage are included in the temporal logic in order to perform control-flow and data-flow testing on BPEL. Zheng et al. also propose an automata called Web Service Automaton (WSA) [391] that is used to transform BPEL into Promela for SPIN, or SMV for NuSMV model-checker. WSA aims to include BPEL data dependencies that cannot be represented in other automata-based formalisms. This approach generates test cases using counterexamples to perform conformance tests on BPEL and using WSDL to test web service operations.

Huang et al. [141] propose 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 (PDDL) for use with the BLAST model checker. Using BLAST, negative and positive test cases can also be generated. Huang et al. propose an extension to the OWL-S specifications and the PDDL in order to support this approach and use a modified version of the BLAST tool.

Gao and Li [115] introduce a probabilistic timed interface automaton for BPEL4WS compositions called PTIA4WS. The proposed automaton includes stochastic and time-related behaviour of the composition which are not included in BPEL. Similar to the previous approaches test cases are generated using from counter examples. The authors claim that due to the extra behavioural information the proposed model allow generation of time and probability optimal test cases.

Zhao et al. [389] propose an approach can both verify and validate BPEL processes using model checking. The proposed approach transform BPEL to LOTOS in order to perform model checking with EVALUATOR (v3.0). After, verification EVALUATOR tool provides an LTS which is converted to a TTCN behaviour tree to validate BPEL behaviour.

Jokhio et al. [154] propose 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 using the assertion violation property of the ProB model checker.

Qi et al. [263] claim that existing software model-checkers cannot verify liveness in real code and propose an approach that aims to find safety and liveliness violations in ScS. The approach uses the Finite Trace LTL Model Checking (FTLT-MC) tool to determine whether a SUT satisfies a set of safety and liveness properties. The authors also claim that this is the first approach for C++ web services.

Betin-Can and Bultan [32] propose the use of model-checking to verify the interoperability of web services. In this approach, it is assumed that the peers of a composite web service are modelled using a Hierarchical State Machine (HSM) model. Betin-Can and Bultan propose a modular verification approach using Java PathFinder (JPF) [150], a Java model checker, to perform interface verification, SPIN for behaviour verification and synchronizability analysis.

Ramsokul and Sowmya [264] propose the modelling and verification of web service protocols via model-checking. Ramsokul and Sowmya propose a distributed modelling language based on the novel Asynchronous Extended Hierarchical Automata (ASEHA), which is designed for modelling functional aspects of the web service protocols. ASEHA model of web service protocols are translated into Promela and correctness of the protocol is verified by the SPIN tool.

Guermouche and Godart [126] propose 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. [378] propose an approach for verifying multi-business interactions. In the approach, business processes are formalised as Pi-Calculus expressions, which are then translated into SMV input code in order to use with SMV model-checker.

2.6.2.4 Model-Based Testing & Verification Using Petri Nets

Petri Nets are widely used for specifying and analysing concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic systems [226]. Petri Nets allow different analyses on the model such as reachability, boundedness, deadlock, liveness, reversibility, fairness and conservation analysis. Petri Nets can also be used for measuring test case coverage.

Petri Nets have also been used in model-based testing of web services. For example, Dong and Yu [87] propose a Petri Net based testing approach where 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 datatypes and policy-based constraints specified by the tester provide the necessary constraints for test data generation.

Wang et al. [338] propose 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. [241], who proposed a method for transforming BPEL processes into Petri Nets with formal analysis of BPEL processes using Petri Nets models. Two tools are used to automate the transformation and analysis; BPEL2PNML [356] and WofBPEL [356]. BPEL2PNML is a tool that generates the Petri Net model and WofBPEL is a tool that performs static analysis on Petri Net models. WofBPEL is capable of checking for unreachable BPEL activities and competition problems for inbound messages.

Schlingloff et al. [282] propose a Petri Nets-based model-checking approach using the Lola model-checking tool [283] to verify BPEL processes. Schlingloff et al. introduced a usability analysis to verify the expected behaviour of participating web services.

Lohmann et al. [194] address the problem of analysing the interaction between BPEL processes, using a special class of Petri Nets called open WorkFlow Net (oWFN). Lohmann et al. introduce two tools that support this approach called BPEL2oWFN that transforms BPEL to oWFN or Petri Net. oWFN model is used by another tool called Fiona that analyses the interaction behaviour of oWFNs. Petri Net models are used with model checkers to verify the internal behaviour of a BPEL process.

Moser et al. [225] propose a method that increases the precision of Petri Net based verification techniques. The authors claim that most of the existing verification methods neglect data aspects of BPEL processes. The authors highlight that their approach incorporates data dependencies into analysis. Data flow is extracted from BPEL by creating a Concurrent Single Static Assignment Form (CSSA). Important data items in CSSA are identified and mapped into the Petri Net model.

Different flavours of Petri Nets are also used in modelling BPEL due to their more expressive nature. For example, Yang et al.'s [368] approach is one of the earliest works that proposes the use of Colored Petri Nets (CP-Nets) [151], an extended Petri Net formalism, for modelling BPEL processes. Yang et al. list the capabilities of CP-Nets that allow different levels of verifications of BPEL processes such as reachability, boundness, dead transition, dead marking, liveness, home, fairness and conservation analysis. The proposed framework uses CPNTools [75] for CP-Nets analysis and can also verify the BPEL to CP-Net transformation.

Yi et al. [371] also propose a BPEL verification framework that uses CP-Nets. Yi et al. claim that CP-Nets are more expressive than FSM and Petri Nets, and propose the use of CP-Nets in order to model the specifications of the web service conversation protocols. The proposed framework can also help with composition of new BPEL processes and verify the existing processes.

Dong et al. [88] propose the use of HPNs. The proposed approach uses a modified version of a tool called Poses++, which is also developed by the authors. The tool is used for automated translation from BPEL to HPN and is also capable of generating test cases.

Dai et al. [78] propose 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. claim that these annotations can be used in verification of non-functional properties of BPEL as well. This is supported by a tool called MCT4WS that allows automated verifications for web service compositions.

Xu et al. [363] propose the use of the Synchronized-Net model, a model based on Petri Nets. For this approach, they use a transformation tool called BPEL2PNML, capable of transforming BPEL into Petri Net Markup Language (PNML) [256]. 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. [180] propose a model-based approach for test coverage analysis. The proposed approach uses Resource Description Framework (RDF) [269], a language for representing information about resources in the WWW, for specifying the preconditions and effects of each method in WSDL specifications. The preconditions and effects specify, respectively, the state of the service before and after invoking a method.

Yang et al. [367] propose 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 claim that incorporating the effects of the BPEL's dead-path-elimination into static defect analysis reduces both false positives and negatives.

Felderer et al. [106, 107] propose a model-driven testing framework called Telling TestStories (TTS). TTS enables tests-driven requirements testing for ScS. Felderer et al. introduce a domain specific language that allows formalisation of system requirements and test model and test cases to be specified based on the concepts of the requirements. Two models are defined in the TTS framework: the system model which describes system requirements at business level and the test model that contains test case specifications. The TTS framework can also ensure the quality of the test artefacts.

Frantzen et al. [111] propose a model-driven development and testing platform called PLASTIC[260]. PLASTIC is supported by two tools that help model-based testing called JAMBITON and MINERVA. These two tools provide support for off-line functional validation for services. In this approach, a Symbolic Transition System (STS) representation of services is provided by the provider. JAMBITON can automatically test services using the provided STS. The authors claim that JAMBITON is the only tool that can provide automated on-the-fly testing.

2.6.3 Experimental Results

Huang et al. [141] experimented on a synthetic online shopping system. The authors generated 7 positive test cases using BLAST and 9 negative test cases using the so-called Swiss-Cheese approach [326] with 100% pass rate.

Jokhio et al. [154] 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 (3 test cases) and modified condition decision coverage (30 test cases).

Paradkar et al.'s [252] automated conformance tests generation approach is validated using a real industrial application with 4 web services and 84 operations. The authors achieved 100% requirement coverage during the their experiments with a stable version of the application. During experiments, they revealed 4 functional bugs out of 7 bugs known to be present. The authors also compared the effort and efficiency of the approach, demonstrating that the approach can reduce the cost around 50% compared to manual testing used by the testing team of the company.

Endo et al. [98] experimented on three different examples: CGD, 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 to 10 test cases.

Li et al. [179] experimented with a real web service that wraps query function of a call centre. They generated test cases for 23 out of 128 input fields of the selected service. The authors' approach generated 514 test cases within 10 minutes using datatype constraints and boundary information. The authors highlighted the fact that this approach reduces cost vastly compared to 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 [23] experimented on a real-word 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. 6 of the revealed faults are found out to be severe.

Endo et al. [97] experimented on an example banking system. The proposed approach achieved 100% coverage using 6 positive test cases in all requirements (all-nodes, all-edges, all-uses and all-Pot-uses). 19 negative test cases are needed in order to achieve 100% coverage on the faulty pairs of ESG. Test case minimisation capability of the proposed approach is also evaluated and the approach achieved between 74% to 64% reduction in test suit size.

Yan et al. [365] 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. [138] performed mutation analysis on six different BPEL programs: 2 versions of loan approval, 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 [65] experimented on a real RESTful service. The authors performed a daily testing routine which took 5 minutes to execute the 300 test cases. 5 to 10 test cases failed daily out of 300 cases. The authors also automatically generated a test suite with 42,767 test cases covering all possible input parameters. 38,016 test cases initially failed and in the second attempt after bug fixes, 1781 test cases still failed.

Blanco et al.'s [37] search-based test data generation approach was run on two example BPEL programs: loan approval and shipping service. Blanco et al.'s approach 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. [63] evaluated model-based testing 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 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. [180] experimented on two web services: the Parlay X Conference web service and the CSTA Routing Service. The authors performed their experiments on a machine with a 3GHz processor and 2GB of memory. The flow analysis of Parlay X service with 9 operations took 3.00 seconds and the CSTA service with 10 operations took 10.00 seconds.

García-Fanjul et al. [117] experimented within a loan approval example BPEL program. The authors claim that the approach is capable of finding the 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 3.0GHz Pentium4 processor and 2GB of memory. The model that was used in the experiments had 32 internal states and was represented by a 96 bytes state-vector.

2.6.4 Discussion

MBT 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 [114, 117, 141, 392] in testing is their ability to generate test cases. These approaches, combined with simulated testing of BPEL compositions, discussed in Section 2.4, 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 chapter to retain a focus on testing and verification.

Model-based test case generation is a well-known and exercised technique. Since 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 Endo et al. [98] and Blanco et al.'s [37], where required number of test cases for coverage are much fewer than random, prove that they reduce the cost of testing not just 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 [37, 98, 125, 138, 167, 168, 365, 379] described above 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 above 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 Felderer et al. [106] require the test model to be generated separately.

2.7 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 [342] 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. [286] describe the possible interoperability issues regarding core web service specifications such as SOAP, WSDL and UDDI and explain how the WS-I Basic Profile provides solutions for the interoperability issues with web service specifications.

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 which use a certain toolkit might cause interoperability problems (due to message consumption errors) for the clients using other toolkits [293]. 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.

2.7.1 Perspectives in Interoperability Testing

Since 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 Narita et al. [227] and Yu et al. [377] that target service compositions. These approaches can only be performed by the integrator.

2.7.2 Interoperability Testing Approaches

The need for testing the interoperability among services is also recognized by researchers. For example, Bertolino and Polini [30] propose a framework that tests the service interoperability using service's WSDL file along with a Protocol State Machine Diagram (PSM) [258] provided by the service provider. The PSM diagram carries information on the order of the operation invocations for a service. The proposed framework checks the order of the web service invocations among different web services and attempts to point out possible interoperability problems.

Yu et al. [377] propose an ontology-based interoperability testing approach using the communication data among web services. The proposed approach captures communication data and stores it in an ontology library. The data in the library is analysed, and reasoning rules for error analysis and communication data are generated in order to run with the JESS reasoning framework [152]. 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 web services.

Smythe [294] discusses the benefits of the model-driven approach in the SOA context and proposes an approach that can verify interoperability of services using UML models. The author points out the need for UML-profile for SOA that contains the interoperability specification in order to use with the proposed approach. Using the proposed the UML-profile, test cases can be generated for testing the conformance of the web service's interoperability.

Similarly, Bertolino et al. [30] propose a model-based approach for testing interoperability. In the proposed environment, web services are tested before registration. In this approach, service provider is expected to provide information on how to invoke a web service using an UML 2.0 behaviour diagram.

Ramsokul and Sowmya [265] propose the use of ASEHA framework to verify the service protocol's implementation against its specification. They claim that the ASEHA framework is capable of modelling complex protocols, such as Web Services Atomic Transaction (WS-AtomicTransaction) [347] and Web Services Business Activity (WS-BusinessActivity) [348]. 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 [126] propose 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 [32] propose the use of a model, based on HSMs, for specifying the behavioural interfaces of participating services in a composite service. Betin-Can and Bultan suggest that verification of the web services that are developed using the Peer Controller Pattern is easier to automate, and propose the use of HSMs as the interface identifiers for web services in order to achieve interoperability. Betin-Can and Bultan propose a modular verification approach using Java PathFinder to perform interface verification, and SPIN for behaviour verification and synchronizability analysis. The use of the proposed approach improves the efficiency of the interface verification significantly as claimed by the Betin-Can and Bultan.

Narita et al. [227] propose a framework for interoperability testing to verify web service protocols, especially aimed at reliable messaging protocols. Narita et al. claim that none of the existing testing tools aim to perform interoperability testing for communication protocols. They also highlight 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 web services.

Passive testing is the process monitoring the behaviour of the SUT without predefining the input(s) [62]. The first passive testing approach for web services is proposed by Benharref et al. [25]. 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 WSUT in order to speed up the state recognition and variable assignment procedures.

Passive conformance testing of EFSMs was proposed by Cavalli et al. [62], where testing artefacts called ‘invariants’ enable testing for conformance. These invariants contain information on the expected behaviour of the SUT and used in testing traces. Several authors extended this research to web services domain.

For example, Andrés et al. [5] propose the use of passive testing for service compositions. The proposed invariants contain information on 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. [222] propose a set 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. [53] also propose 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 present a tool that automates the passive testing for behavioural conformance called RV4WS. Cao et al. [50] extended this approach by integrating the active testing tool WSOTF.

Yu et al. [375] investigated the number of requests required to adequately test the interoperability of geospatial web services. In the work, the authors proposed a specification-based service request models to help generate the requests for testing.

2.7.3 Experimental Results

Narita et al. [227] performed experiments on Reliable Messaging for Grid Services (RM4GS) 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 message order and sending duplicate messages. During coverage testing, the framework tested 180 out of 212 WS-Reliability items. It was unable to find any errors but raised 3 warnings. During application-driven testing 4 errors were revealed and 4 warnings were raised.

Betin-Can and Bultan [32] experimented on the travel agency and the order handling examples. For the travel agency, different peers of the program took between 4.61 seconds to 9.72 seconds for interface verification. The resources used in this experiment is ranged from 3.95MB and 19.69MB of memory. Synchronizability analysis of the travel agency (which was 8,911 states) took 0.38 seconds and used 5.15 MB of memory. Order handling interface verification for peers took between 4.63 to 5.00 seconds and used 3.73MB to 7.69MB of memory. Synchronizability analysis of order handling (which was 1,562 states) took 0.08 seconds and used 2.01MB of memory.

2.7.4 Discussion

As stated, interoperability is one of the major strengths 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 requires approaches, such as Tsai et al. [323] and Bertolino et al. [30], where interoperability is tested before service registration. Testing web services before the registration process can prevent many of the possible interoperability problems and this could increase 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. [227] and Ramsokul and Sowmya [265]. The second group verify interfaces and communication among services, such as the work of Betin-Can and Bultan [32], Smythe [294] and Yu et al. [377]. The third group are the passive testing approaches, such as the work of Andrés et al. [5], Morales et al. [222] and Cao et al. [53].

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 with the 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 due to the effort required to create the necessary invariants.

2.8 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. Since 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. By performing integration testing, all the elements of a ScS can be tested including services, messages, interfaces, and the overall composition.

Bendetto [24] defined the difference between integration testing of traditional systems and ScS. Canfora and Di Penta [48] point out the challenges in integration testing in SOA. According to Bendetto and Canfora & Di Penta, the challenges of integration testing in ScS are:

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 due to the number of required service calls.

2.8.1 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 due to the lack of observability.

Most of the approaches in this section target service compositions using static binding. In contrast to dynamic SOA, performing integration testing can be very challenging due to 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.

2.8.2 Integration Testing Approaches

One of the earliest works on integration testing of web services is Tsai et al.'s [325] Coyote framework. Coyote is an XML-based object-oriented testing framework that can perform integration testing. Coyote is formed of two main components: a test master 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. On the other hand, the test engine performs the tests and logs the results for these tests.

In software development, there is a concept called Continuous Integration (CI) [90]. 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. [142] propose a simulation framework that addresses the service availability problem using CIT. The proposed framework automates the testing by using a surrogate generator, that generates platform specific code skeleton from service specifications and a surrogate engine that simulates the component behaviour according to skeleton code. Huang et al. claim that the proposed surrogates are more flexible than the common simulation methods, such as stubs and mocks, and the simulation is platform-independent.

Liu et al. [192] also propose a CIT approach in 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 introduce a test execution engine to support this approach.

Peyton et al. [257] propose a testing framework that can perform ”grey-box” integration testing of composite applications and their underlying services. The proposed framework is implemented in TTCN-3 [102], an European Telecommunications Standards Institute standard test specification and implementation language. It is capable of testing the composite application behaviour and interaction between participating web services. The framework increases the visibility and the control in testing by inserting test agents into a web service composition. These agents are used in analysing HTTP and SOAP messages between the participating services.

Mei et al. [214] address the integration issues that might be caused by XPath in BPEL processes, such as extracting wrong data from an XML message. The proposed approach uses CFGs of BPEL processes, along with another graph called XPath Rewriting Graph (XRG) that models XPath conceptually (models how XPath can be rewritten). Mei et al. created a model that combines these two graphs called X-WSBPEL.

Data-flow testing criteria based on def-use associations in XRG are defined by Mei et al. and using these criteria, data-flow testing can be performed on the X-WSBPEL model.

Sun et al. [307] introduces an approach that adapts popular fault localisation Tarantula [155] aiming to discover integration-level faults in BPEL compositions. The approach focuses on faults related to interactions which coordinate service invocations. In the proposed approach, Tatantula method is applied to the execution traces of the test suite (executed blocks) in order to discover the suspicious blocks.

De Angelis et al. [80] propose 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 present a tool that supports this approach.

Tarhini et al. [312] address the issue of web service availability and the cost of testing. They solve 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 web services from a service broker are found. In the second stage, selected web services are tested for their correct functionality. At the third stage, each web service is tested for interaction as a stand-alone component and, if it passes this stage, it is tested for interactions with the rest of the components. When a web service passes all the required steps it is logged into the list of services to be invoked at runtime. The proposed framework uses a modified version of the Coyote framework for the automation of testing.

Garriga et al. [118] propose the inclusion of integration testing in the service selection process in SOA. In this process discovered candidate services are selected based on the results of a testing process. The approach tests services for interface and behavioural compatibility. Similarly, Miao and Liu [220] also propose the inclusion of a testing phase in the service selection process. In the proposed approach test cases and sequences are generated functional scenarios. These scenarios are generated from user requirements which include pre- and pos-conditions for service operations.

Yu et al. [376] address the interaction problems within OWL-S compositions. Yu et al.'s approach tests interaction among participating web services using interaction requirements. Yu et al. propose an extension to existing OWL-S models to carry these requirements.

There are also previously mentioned approaches that are capable of performing integration testing. For example, Tsai et al.'s ASTRAR framework [321] and the proposed Enhanced UDDI server [323] are also capable of performing integration testing. Similarly, Lenz et al.'s [176] model-driven testing approach can be used for integration testing. Sasikaladevi and Arockiam [281] propose a integration testing framework that is capable of generating test cases from WSDL specifications and simulate service clients.

2.8.3 Experimental Results

Huang et al. [142] experimented on a human resources system that is transformed into a web service. In this system, there are 17 components in the business layer with 58 interfaces and 22 components in data layer with 22 interfaces. During the pure simulation without real components, 7 bugs are identified caused by issues such as reference to a wrong service, interface mismatch and missing service. During real component tests (includes surrogates as well), 3 bugs are identified for five components.

Mei et al. [214] 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% to 67% whereas random only achieved 18%. Mean rates of fault detection rates for this approach are between 92% to 98% whereas random achieved 73%. The authors also investigated the performance of the approach. It took between 0.45 to 1.2 second for generating test sets with a 2.4 GHz processor and 512MB memory.

Liu et al. [192] experimented on two synthetic examples: an HR system and a meeting room management system. The approach revealed 22 bugs in the HR system and 7 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.

2.8.4 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 Tarhini et al. [312], Huang et al. [142] and Liu et al.'s [192] frameworks.

Huang et al.'s [142] CI based integration testing can be very useful by starting testing early. The ability to use surrogate services can also help reduce the cost of testing. Since 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.

Liu et al. [192] partly automate test case generation in CIT using sequence diagrams. This approach makes use of Huang et al.'s work and is able to simulate unavailable components. As a result, it has the same restrictions as Huang's work 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.

Mei et al.'s [214] approach 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 prove the effectiveness of their approach in revealing these problems.

Almost all of the approaches discussed above will have problems adapting to dynamic environments. For example, Tarhini et al.'s [312] approach might be rendered inapplicable due to 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, Huang et al. [142], Peyton et al. [257], Tsai et al. [325] and Mei et al.'s [214] approaches might become more expensive to perform.

2.9 Collaborative Testing of Service-centric Systems

Collaborative software testing is the testing concept where multiple stakeholders involved in a web service, 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 are identified by Canfora and Di Penta [48], some of which require collaborative solutions. These challenges that might require collaborative solutions are:

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

2.9.1 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.

Palacios et al. [246] surveyed existing approaches and identified four common stakeholders in approaches that suggest collaboration: provider, registry, third-party and client. In this section, we only mention three stakeholders (developer, third-party and integrator) as we accept the developer, the provider and the registry as a single stakeholder (the developer) since the test cases provided by all these three stakeholders are generated by the developer.

2.9.2 Collaborative Testing Approaches

Tsai. et al. [318] propose a Co-operative Validation and Verification (CV&V) model that addresses these challenges instead of the traditional Independent Validation and Verification (IV&V). One example of this collaborative testing approach is Tsai et al.'s proposed enhanced UDDI server [323]. This UDDI server further enhances the verification enhancements in UDDI version 3 [329]. These proposed enhancements include:

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

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

Bai et al. [18] also propose a contract-based collaborative testing approach that extends Tsai et al.'s enhanced UDDI proposal. Bai et al. propose a Decentralized Collaborative Validation and Verification (DCV&V) framework with contracts. The proposed framework consists of distributed test brokers that handle a specific part of the testing process. Bai et al. suggest two types of contracts for the DCV&V approach: Test Collaboration Contracts (TCC) that enforce the collaboration among the test brokers and Testing Service Contract (TSC) that is used for contract-based test case generation.

Bai et al.'s proposed test broker provides a test case repository for test cases and test scripts, collects test results, maintains defect reports and web service evaluations. The test broker can generate and execute test cases as well. Bai et al. suggest that by using 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 [388, 395, 396] proposes another collaborative approach. In Zhu's approach, service developers or third stakeholder testers provide testing services that help with testing. Zhu also proposes 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. [19] introduce a collaborative testing approach that "whitens" the ScST by introducing a new stakeholder called TCov that provides the tester with coverage information. Bertolini et al.'s approach requires the service provider to insert instrumentation code inside the service in order to provide TCov with coverage information. This information is then analysed by TCov provider and is made available to the tester as a service.

Eler et al. [96] propose an approach to improve web service testability. The proposed approach provides the tester with a 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 present a web service that automatically generates the testing service from Java byte code called JaBUTiWS and also another tool called WSMTS, that support the testing process.

El Ioini and Sillitti [94, 95] propose a collaborative testing framework that allow collaboration among service integrators. The proposed framework acts as a test case repository where test cases submitted by integrators and their execution traces are stored. In the proposed approach test cases generated automatically generated from tester provided specifications and TestGen4J libraries.

2.9.3 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.

Tsai et al.'s [323] proposed testing approach 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 web 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 web services.

Bai et al.'s [18] contracts 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. Bai et al.'s contracts additionally enforce collaboration among stakeholders. Through contract-based collaboration enforcement may be effective, however, the cost of generating contracts might be discouraging.

The main advantage of Zhu's [388, 395] proposed testing environment is that testing can be fully automated. Another advantage is that the SUT is not affected by the testing process. As a result, 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 stakeholder testing services which require no testing. Using third stakeholder testing services might increase the cost vastly due to the costs of testing services.

Bartolini et al. [19] provide experimental results from application of their TCov environment. The authors highlight an important aspect of using TCov is that it enables the tester to receive information on test coverage without violating the SOA principles. The authors also suggest 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 service 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. [96] 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 web service and compared the execution time of the same test suite with the testing service. The overhead added by the approach increased execution times by 2.65% outside the testing session and 5.26% in the testing session. This approach provides the benefits of Zhu's [395, 388] approach and Bartolini et al.'s [19] 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.

2.10 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. [47] 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 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 [350].

In the literature, QoS generally refers to non-functional 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. [173]. 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 two main questions:

1. The selection of the right service.

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. Web 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 Web services and the problems surrounding it has led to the service standard called WS-Agreement [346]. WS-Agreement is a specification language aimed at standardising the overall agreement structure. The language specifies domain-independent elements that can be extended to any specific concept.

Due to 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.

2.10.1 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 to be a static worst-case execution time analysis using the non-functional 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 the 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.

2.10.2 QoS Testing Approaches

The oldest research publication on QoS testing of services is by Chandrasekaran et al. [68]. The authors propose an approach that is used to test performance of web services and simulation based evaluation for service compositions. The authors introduce their tool called Service Composition and Execution Tool (SCET) which allows service composition as well as evaluation. Simulation in the tool is handled by JSIM, a Java-based simulation environment.

Di Penta et al. [83] propose testing for SLAs violations using search-based methods. In this approach, inputs and bindings for ScS are generated using genetic algorithms (GA). 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 which then 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.

Di Penta et al.'s [82] regression testing approach aims to test non-functional 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 [124] propose another search-based SLA testing approach. Similar to Di Penta et al.'s approach, 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. [245] propose a partition-testing based SLA testing approach. In the proposed approach, test specifications are generated from information in SLAs which 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. [28] suggest that test-beds must also be able to evaluate QoS properties in SOA. The authors propose a test-bed that allows functional and non-functional 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) [27]. 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 generation of stubs with both functional and non-functional properties.

Another example to test-beds that can perform offline testing is Grundy et al.'s [123] MaramaMTE [206] tool. This tool provides similar functionality to Bertolino et al.'s test-bed. 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 BPMN and ViTaBal-WS. Unfortunately this approach does not use SLAs to inform the tester about the possible violations.

Driss et al. [89] propose 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 propose a model that includes BPEL activities and network infrastructure. Simulations in this approach are performed using the NS-2 simulator.

Pretre et al. [261] propose a QoS assessment framework using model-based testing 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 is generated from the model. The framework uses a categorisation of tests, goals and results. In order to automate categorisation process, requirements in the form of OCL are provided. Total automation is one of the highlights of this approach.

Yeom et al. [370] introduce 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 web services. These interfaces provide functions to get internal information about services to increase observability or to provide change notifications.

2.10.3 Experimental Results

Di Penta et al. [83] 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 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. [82] experimented on synthetically generated web 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 that SLAs might be violated by the new versions if they cover the newer versions.

Gu and Ge [124] performed their experiments on a synthetic service composition. Their experiments showed that their approach was able to identify QoS risky-paths and generate test cases that can violate QoS constraints. They have also compared their GA approach to random search similar to Di Penta et al. and according to their results GA outperformed the random search.

Driss et al. [89] experimented on a travel planner composite service example. The authors focused 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.

2.10.4 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 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 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. Since QoS testing can be very expensive, using stubs is an ideal way to reduce cost. Reducing cost is invaluable for the integrator allowing him 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. Driss et al.'s [89] approach 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 runtime SLA violations. The weaknesses of testing with real services is the need to test for all possible bindings and the high costs involved. Di Penta et al. [83] and Gu and Ge's [124] similar approaches can help reduce the cost of testing for SLA violations by detecting high risk paths and focusing testing on these parts of service compositions.

2.11 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 done in a white-box manner [374]. Performing white-box regression testing helps mainly with test case management.

A number of the common test case management and prioritisation methods such as the symbolic execution approach and the dynamic slicing-based approach require testers' access to the source code [374]. However, approaches like the Graph Walk Approach (GWA) by Rothermel and Harrold [271] that does not require access to the source code, can be used in ScS Regression Testing (ScSRT). This chapter 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. Since 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 the literature [48, 230], one of the main issues in ScSRT at the consumer side is not knowing when to perform regression testing. Since the consumer has no control over the evolution of the web service, he might not be aware of the changes to the web 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 v3 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 web 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 web 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 web 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, confidence in the service might decrease due to errors or decreases in QoS.

Another challenge in ScSRT is the concurrency issues that might arise during testing due to the tester not having control over all participating web services. Ruth and Tu [275] 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 localisation. 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.

2.11.1 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 Ruth and Tu [276], Liu et al. [191], Mei et al. [213] and Tarhini et al. [310] are aimed to be performed by the integrator. Approaches such as Tsai et al. [327], Di Penta et al. [82] and Hou et al. [139] are expected to be performed by the integrator and the certifier. Lin et al.'s [188] approach can only be performed by the developer.

2.11.2 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 Control Flow Graph (CFG) generation.

Ruth and Tu [276] propose a regression testing approach that is based on Rothermel and Harrold's GWA technique. This approach assumes that the CFGs of participating services are provided by their developers. Ruth and Tu also propose that the test cases and a table of test cases' coverage information over the CFG must also be provided along with WSDL file via WS-Metadata Exchange Framework [351]. 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. [274] also propose 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* [276]. 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 localisation. In this strategy, if a fault is found in a downstream service, none of the upstream services are tested until the fault is fixed. Ruth et al. also take the situation into consideration where a service makes multiple calls to different services in parallel.

Lin et al. [188] propose another GWA based regression testing approach where CFGs are created from Java Interclass Graph (JIG) [135]. A framework that performs RTS on the transformed code of a Java based web service is also proposed by Lin et al. The code transformation can be performed only in Apache Axis framework [6]. The proposed approach uses the built-in WSDL2Java [357] generated 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 JIG on which the GWA can be performed. Compared to the previously presented approaches, Lin et al.'s approach 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. [191] address the issues that occur due to concurrency in BPEL regression testing. Lui et al. propose a test case selection technique based on impact analysis. The impact analysis is performed by identifying the changes to the process under test and discovering impacted paths by these changes.

Tarhini et al. [310] propose another model-based regression testing approach using the previously explained model in Section 2.6. The proposed model is capable of representing three types of modifications to the composite services:

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

The changes that are made to the system are represented in the modified version of the original TLTS. The second and third type 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 is proposed by Wang et al. [335]. This approach uses the BFG [379] that was described in Section 2.6.2.1. Wang et al. propose a BPEL regression testing framework that can generate and select test cases for regression testing using Rothermel and Harrold's RTS technique. Wang et al. extended the BFG model into another graph called eXtensible BFG (XBFG) that the authors claim is better suited to regression testing. Li et al. [177] introduce another XBFG based ScSRT approach where test case generation and selection is based on the comparison of different versions of BPEL applications. Yang et al. [366] propose an approach that aims at providing effective fault localisation 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. [213] propose a different black-box test case prioritisation technique for testing web service compositions. In this approach, test case prioritisation is based on the coverage of WSDL tags in XML Schemas for input and output message types. Askarunisa et al. [12] also proposed a black-box prioritisation approach for OWL-S services. The proposed approach provides prioritisation based on several criteria such as fault rate, fault severity and number of activities or transitions covered.

Mei et al. [217] also propose another coverage model which captures BPEL process, XPath and WSDL that enables test case prioritisation. 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 [13] propose a model-based test case prioritisation approach for service compositions. The approach discovers the most important activity paths using a UML activity diagram of the service composition under test.

Chen et al. [71] also propose a model-based test case prioritisation approach based on impact analysis of BPEL processes. The authors introduce a model called BPEL Flow Graph (BPFG) into which BPEL processes are translated for change impact analysis. Test cases are prioritised according to the proposed weighted dependence propagation model.

Zhai et al. [383] propose a test case prioritisation technique that incorporates service selection for location-aware service compositions. In order to overcome the potential issues caused by dynamic service selection, the authors propose 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 introduce a concept called Point Of Interest (POI) aware prioritisation technique that is more effective than traditional input directed techniques for location-aware services.

Nguyen et al. [231] propose a test case prioritisation approach for “audit testing” of web services. Audit testing aims to validate the interoperability of new versions of the invoked services. The proposed prioritisation approach determines change sensitivity of each test case and uses this metric to prioritise them. Change sensitivity is measured using a mutation analysis where the execution of monitored service responses are compared to the execution of their mutants. Mutant responses are generated by applying the nine mutation operators (introduced by the authors) to the monitored service responses.

Nguyen et al. [229] also proposed another prioritisation approach using past execution traces and Information Retrieval (IR). In the approach past execution traces are used to generate test case identifier documents which contain terms that appear in method signatures, variable and class names. Then, IR is applied to search for test cases that are related to the given service change description (by the provider). The ranking of the identifier documents provided by IR tools is used for test case prioritisation.

Mei et al. [216] propose a regression testing approach that addresses the issues regarding late service binding in service compositions called Preemptive Regression Testing (PRT). The approach is aimed at regression testing of services that can change processing logic or invoke external services during their execution. During the curse of regression testing when a change detected PRT takes control of the regression testing and searches for test cases (in the current test suite) that cover the recently modified part of the service. If test case(s) covering the modified part is found these test case(s) first get executed then the regression testing continues if there is no more modifications are found.

Zhang et al. [385] propose a regression testing model called WSRTM aiming to test semantic web services. The model captures help identifying changes to the WSDL ad IOPEs of the service. Based on the generated model and testers coverage choice, test cases are generated for regression testing. Generated test cases aim to validate the sequential functional behaviour of services.

Srikanth and Cohen [301] propose a regression testing approach for SaaS environment. In the approach use cases of previous failures are modelled as abstract events. Then, event sequences are created using these models and the tester provided constraints according to the testers coverage criteria selection.

The need for a visual testing tool is addressed by Pautasso [254]. The author proposes a framework called JOpera. JOpera is capable of performing unit and regression testing on web service compositions. The proposed tool is implemented using a language called JOpera Visual Composition Language [255]. 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 localisation. Chaturvedi [69] also propose a framework automates the regression testing of web services. The tool generates a graph of the web service (based on WSDL) which is used to determine the modified parts of the code.

Tsai et al. [327] propose a Model-based Adaptive Test (MAT) case selection approach that can be applied to both regression testing and group testing. This approach defines a model called the Coverage Relationship Model (CRM) that is used for test case ranking and selection. Using this model, test cases with similar aspects and coverage can be eliminated. Tsai et al. define multiple rules that guarantee the selection of the most potent test cases and prove that the less potent test cases never cover the more potent test cases. Tsai et al. claim that this approach can be applied to regression testing when a new version of a service with the same specifications is created.

In ScSRT, test case management has other test case prioritisation considerations such as service access quotas. Hou et al. [139] address 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. [139] propose a scheduled regression testing that divides the testing process into several parts according to the user's access quotas, while ignoring the actual execution time of the regression testing. The aim of this approach is to divide test cases into groups based on time slots that suit the user's web service access quotas. The proposed test case prioritisation approach is based on a multi-objective selection technique, which defines an objective function that aims to attain maximum coverage within the quota constraints of services.

Di Penta et al. [82] propose a collaborative regression testing approach that aims to test both functional and non-functional properties of web services. Di Penta et al.'s approach allows the developer to publish test cases along with services that are used in the initial testing and regression testing. The approach also reduces the cost of regression testing by monitoring service input-output. All these functionalities are provided by a testing tool that supports this approach.

2.11.3 Experimental Results

Mei et al. [213] experimented on eight BPEL examples and randomly generated 1,000 test cases for these programs. Using these test cases, 100 test suites with average 86 test cases were generated. The authors claim that their black-box testing approaches can achieve similar fault detection rates to white-box testing approaches.

Mei et al. [217] also used the same case study to evaluate their coverage based approach. The authors proposed 10 different techniques for prioritisation. All 10 techniques were found to be more effective then random prioritisation. The results also suggest that the techniques augmenting additional coverage information from WSDL elements and XRG provide more effective prioritisation.

Di Penta et al. [82] 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. [383] experimented on a real-world service composition called City Guide. The proposed service selection method reduced service invocations by 53.18%. The POI-aware techniques outperformed the input guided techniques in invocation reduction.

2.11.4 Discussion

The issues that relate to ScSRT for all the stakeholders in SOA are addressed by some of the work discussed above. For example, one of the major issues in ScSRT is that of the integrator not having a realistic test suite. This issue is addressed by Ruth and Tu [276] and Di Penta et al. [82].

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

ScSRT 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 Tsai et al. [327], can help to minimise this problem. Nevertheless, a large regression test suite might require a high number of executions.

An additional concern related to the limitations in performing ScSRT due quota restrictions is addressed by Hou et al. [139]. 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.

2.12 Conclusion

In this chapter we provided a summary of testing techniques and approaches that have been proposed for testing web services. The chapter focused on fundamental functional testing methodologies such as unit testing, regression testing, integration testing and interoperability testing of Web services. The chapter also included testing methodologies such as model-based testing, fault-based testing, formal verification, partition testing, contract-based testing and test case generation. There are two other sections: collaborative testing (testing approaches where multiple stakeholders participate in testing) and QoS violation testing. Each section introduced the relevant approaches, discussed their strengths and weaknesses. Each section also discussed the experimental results that are presented by the authors of each work in order to highlight achievements of these work.

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 this chapter, while others still await effective solutions. Several of the unaddressed issues in ScST need new and more efficient solutions, thus bringing new opportunities and challenges. According to the authors, the current open problems in ScST are:

1. Lack of real-world case-studies.

2. Solutions that can generate realistic test data.

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.

6. Modelling and validation of fully decentralised ScS.

We believe in order to provide a better understanding of the open problems in ScST, the current trends needs to be analysed and discussed. As a result, the following chapter first provides an analysis of the current trends in ScST, and then discusses the open issues in order to predict emerging and possible future trends.

Chapter 3

Analysis of Trends in Service-centric System Testing

3.1 Service-centric System Testing Trends

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

One of the most important factors that proves the significance of a research field is the number of research publications. Figure 3.1 shows the total number of publications from 2002 to 2012. Total number of publications in 2012 might not reflect the actual number due to its recency. In Sharma et al.'s [288] study, the authors identified 146 papers published between 2002 and 2011 in web service testing subject. In our study, for the same period we identified 215 papers.

Analysis of the research publications on the subject revealed its rapid growth. The trend line plotted on Figure 3.1 with coefficient determination value (R^2) $\tilde{0}$.995 is very reassuring to make future predictions. The trend line suggests that if this trend continues until 2014, the number of publications will be around 325, close to twice the number of publications in 2010.

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

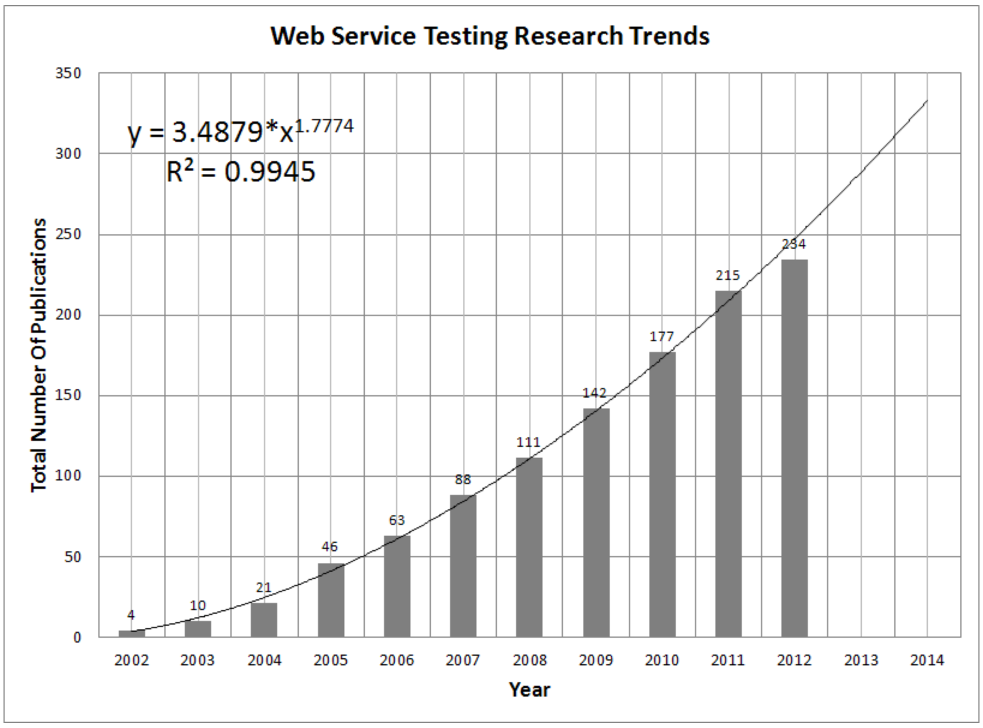


Figure 3.1: Total number of publications from 2002 to 2012

Analysis of the publication volume for the testing techniques and methodologies used in ScST is presented in Figure 3.2 and Figure 3.3. Formal verification, model-based testing and fault-based testing are the three methodologies 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 [39], the history of web service testing is divided into three phases, based on 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, binding capabilities of web services, the asynchronous web service messaging capability and the

SOAP intermediary capability of SOA. Testing for QoS also emerged as a topic of interest during this period.

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

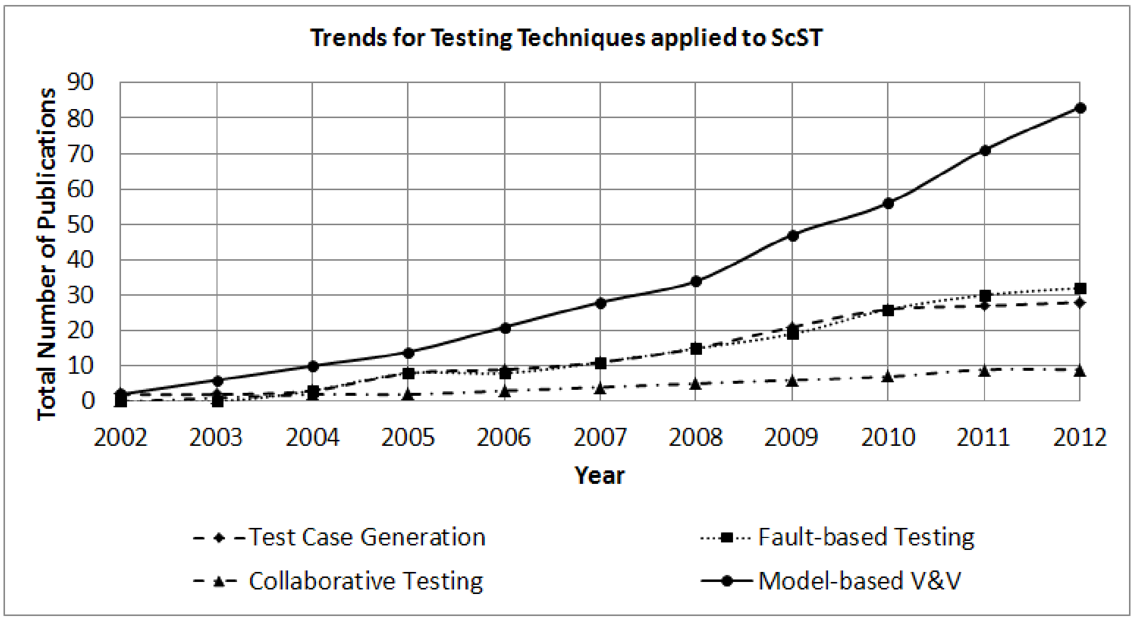


Figure 3.2: Publication trends of testing techniques applied to ScST

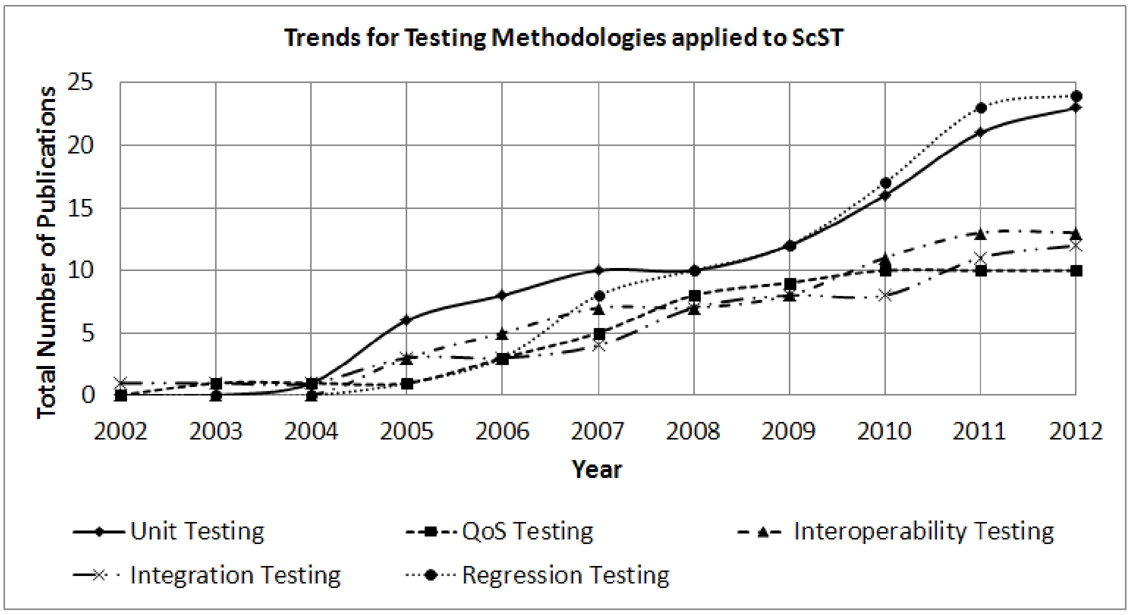


Figure 3.3: Publication trends of testing methodologies applied to ScST

Analysis of the academic work in Figure 3.2 and Figure 3.3 do not yield results similar to Bloomberg's analysis. For example, in the 2002-2003 period, 5 publications on formal verification, 1 on test data generation, 1 on integration, 1 on QoS and 1 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.

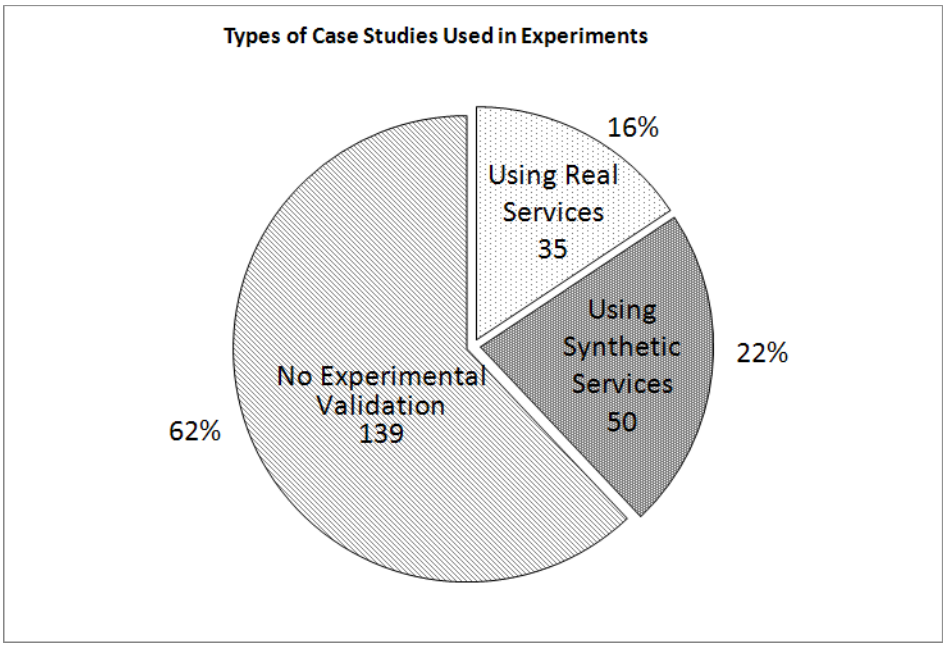


Figure 3.4: Distribution of case study types used in experimental validation. Number inside the circle indicate absolute number of papers.

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 3.4 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, 62% of the research publications discussed in Chapter 2 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 an HR system.

3.2 Categorisation of Testing Approaches Applied to Service-centric System Testing

In this section we provide an overview of the work discussed in Chapter 2. Table 3.1 highlights certain aspects of each work, such as category, web service technologies, testing tools and case studies, making the comparison of the existing work easier. We believe that having a summary of all case studies used in the existing work is invaluable at least for two reasons. First, it helps reduce the effects of the lack of experimental validation in ScST research, which is a major problem. As mentioned in Section 3.1, 71% of the work covered in Chapter 2 do not provide any experimental results. The information presented in Table 3.1 may give the researchers an idea on the available case studies that can be used in their work for experimental validation. Second, having a summary of all used case studies enables researchers to apply their techniques on the same/similar case studies used in the existing work, allowing better comparison among the approaches.

Presenting the summary of the tools used is also important to both researchers and industry. We believe this information enables researchers to discover available tools on a specific testing methodology and use the publicly available ones in their experiments, or for integrating them to their tool/framework. For industry, discovering the state-of-the-art tools developed by research communities might provide opportunities for the application of these approaches to real-world systems.

Table 3.1 also enables researchers to easily notice the gaps in service technologies that are not covered by any of the existing work. The information from the table might bring opportunities as well as questions on the reasons that limit application of the existing approaches to that specific service technology.

Table 3.1: Summary of ScS Testing and Verification Approaches

Author	Testing Category	WS Technology	Tools	Services/ScS Used in Experiments
Bartolini et al. [20]	Specification based test data generation	WSDL	WS-TAXI	Pico service
Ma et al. [201]	Specification-based test data generation	WSDL	–	Synthetic order system (3 versions)
Bai et al. [16]	Specification-based test data generation	WSDL	–	356 public services
Li et al. [185]	Specification-based test data generation	WSDL	WSTD-Gen	Call center query service
Chakrabarti and Kumar [65]	Test data generation	WADL	–	Real RESTful web service
Bozkurt and Harman [42]	Specification-based test data generation	OWL-S	–	15 public services
Paradkar et al. [252]	Model-based testing	OWL-S	–	An online application for asset management
Dong and Yu [87]	Model-based testing	WSDL	–	–
Ma et al. [202]	Model-based testing	BPEL	–	–
Guangquan et al. [125]	Model-based testing	BPEL	–	Book renewal process
Conroy et al. [74]	Test data generation	WSDL	–	Accenture People Directory and University Data applications
Yan et al. [365]	Model-based testing	BPEL	–	Loan approval
Yuan et al. [379]	Model-based testing	BPEL	–	–
Endo et al. [98]	Model-based testing	BPEL	ValiBPEL	GCD, Loan approval and nice journey
Kaschner and Lohmann [164]	Model-based testing	BPEL	–	Example online shop
Hou et al. [138]	Test case generation	BPEL	–	Loan approval(1-2), ATM, market place, gymlocker, BPEL(1-5)
Casado et al. [59]	Model-based testing	WS-COOR, WS-BA	–	Loan approval
Casado et al. [57, 60, 58, 61]	Model-based testing	–	–	–
Blanco et al. [37]	Test case generation using search-based methods	BPEL	–	Loan approval and shipping service

Continued on Next Page…

Table 3.1 – Continued

Author	Testing Category	WS Technology	Tools	Services/ScS Used in Experiments
Heckel and Mariani [137]	Partition testing	WSDL	–	–
Park et al. [253]	Partition testing	WSDL	–	–
Bertolino et al. [29]	Partition testing	WSDL	TAXI	–
Sun et al. [305]	Partition testing	WSDL	–	Synthetic electronic payment system
Bai et al. [17]	Partition testing	OWL-S	–	Synthetic travel system
Heckel and Lochmann [136]	Contract-based testing	WSDL	–	–
Mei and Zhang [212]	Contract-based testing	WSDL	–	2 synthetic services (Tritype and Middle)
Dai et al. [77]	Contract-based testing	OWL-S	CBT4WS	–
Andrés et al. [5]	Contract-based testing	–	–	–
Noikajana and Suwannasart [232]	Contract-based testing	WSDL-S	–	2 synthetic services RectangleType and IncreaseDate
Askarunisa et al. [11]	Contract-based testing	WSDL-S	–	2 real services (converttemp and convertvalue) and 2 synthetic services (shape,complex shape)
Liu et al. [193]	Contract-based testing	SAWSDL	–	Synthetic e-learning service
Saleh et al. [278]	Contract-based testing	–	–	–
Sneed and Huang [295]	Unit testing	WSDL	WSDLTest	eGoverment project with 9 Web Services (WS)
Lenz et al. [176]	Model-driven unit testing	–	–	–
Chan et al. [66]	Unit testing using metamorphic relations	–	–	–
Sun et al. [306]	Unit testing using metamorphic relations	WSDL	–	Synthetic ATM example
Tsai et al. [321, 317]	Unit testing	WSDL	ASTRAR	60 versions of synthetic BBS web service
Tsai et al. [328]	Unit testing	–	–	–
Mayer and Lübke [208]	Unit testing	BPEL	BPELUnit	–
Li et al. [186]	Unit testing	BPEL	–	–
Mani et al. [205]	Unit testing	WSDL	Taukan	–
Palomo-Duarte et al. [247]	Unit testing	BPEL	Taukan	–
Zhu et al. [398]	Unit testing	–	SCENETester	4 synthetic compositions

Continued on Next Page…

Table 3.1 – Continued

Author	Testing Category	WS Technology	Tools	Services/ScS Used in Experiments
Illieva et al. [146, 145]	Unit testing	BPEL	TASSA	–
Reza and Van Gilst [270]	Unit testing	RESTful services	–	–
Chandramohan et al. [67]	Unit testing	–	–	–
Hallé [127, 128]	Unit testing	SOAP, WSDL	–	Synthetic service example
Li et al [182]	Unit testing	–	SOArMetrics	–
Sinha and Paradkar [292]	Model-based testing	WSDL-S	–	–
Bentakouk et al. [26]	Model-based testing	–	–	–
Escobedo et al. [101]	Model-based testing	BPEL	–	–
Fu et al. [114]	Model-checking	BPEL	SPIN	–
García-Fanjul et al. [117]	Model-based testing	BPEL	SPIN	Loan approval example
Zheng et al. [392]	Model-based testing	BPEL	SPIN, NuSMV	–
Huang et al. [141]	Model-based testing	OWL-S	BLAST	Synthetic shopping WS
Endo and Simao [99]	Model-based testing	OWL-S, BPEL, WS-CDL	JStateModelTest	ThirdPartyCall-SOA, QualiPSo-Factory
Jokhio et al. [154]	Model-checking	WSMO	ProB	Amazon E-commerce service
Di Petro et al. [84]	Model-checking	–	SMC4WS, NuSMV	Synthetic BPEL example
Ibrahim and Khalil [144]	Model-checking	BPEL	UPAAL	–
Pacharoen et al. [244]	Model-checking	BPEL	LTSA	–
Fu and Chen [113]	Model-checking	–	–	–
Zhao et al. [389]	Model-checking	BPEL	EVALUATOR	synthetic banking process
Luo et al. [198]	Model-checking	OWL-S	MCTK	BravoAirProcess
Gao and Li [115]	Model-checking	BPEL	–	Synthetic online payment system
Gao et al. [116]	Model-checking	–	PRISM	Synthetic services
Qi et al. [263]	Model-checking	–	FTLT-MC	WS-ReliableMessaging example
Oghabi et al. [236]	Model-checking	OWL-S	PRISM	–
Todica et al. [316]	Model-checking	BPEL	SPIN	–
Zhao et al. [390]	Model-checking	–	–	–
Betin-Can and Bultan [32, 33]	Model-checking	BPEL	JavaPathfinder, SPIN	Synthetic travel agency service and purchase order service
Kacem et al. [161]	Model-checking	BPEL	BPELVT, SPIN	Synthetic BPEL process
Ramsokul and Sowmya [264]	Model-checking	WS protocols	SPIN	WS-AT
Hwang et al. [143]	Model-checking	BPEL, WS-CDL	–	Synthetic shopping process

Continued on Next Page…

Table 3.1 – Continued

Author	Testing Category	WS Technology	Tools	Services/ScS Used in Experiments
Hamel et al. [129]	Model-checking	BPEL	RODIN	–
Lallai et al. [168]	Model-based testing	BPEL	BPEL2IF, TestGen-IF	Oracle example loan Service
Cao et al. [51, 52]	Model-based testing	BPEL	WSOTF, TGSE	Oracle example loan Service
Dong and Yu [87]	Model-based testing	WSDL	–	–
Wang et al. [338]	Model-based testing	OWL-S	TCGen4WS	–
Zahoor et al. [382]	Formal verification using satisfiability solving	BPEL	SAT solver	Shipping service
Papapanagiotou and Fleuriot [248]	Formal verification using linear logic	–	HOL Light	Synthetic home purchase
Ouyang et al [241]	Formal verification using Petri Nets	BPEL	BPEL2PNML, Wof-BPEL	–
Schlingloff et al. [282]	Model-checking using Petri Nets	BPEL	Lola	–
Lohmann et al. [194]	Formal verification Petri Nets	BPEL	BPEL2oWFN, Fiona	Onlineshop example
Yang et al.'s [368]	Formal verification Petri Nets	BPEL, WSCI	CPNTools	–
Yi et al. [371]	Formal verification Petri Nets	BPEL	CPNTools	Airline and travel agency service
Dong et al. [88]	Formal verification Petri Nets	BPEL	Pose++	–
Dai et al. [78]	Formal verification Petri Nets	BPEL	MCT4WS	–
Xu et al. [363]	Formal verification Petri Nets	BPEL	BPEL2PNML	–
Moser et al. [225]	Formal verification using Petri Nets	BPEL	–	–
Li et al. [183]	Formal verification Petri Nets	–	–	–
Zhu et al. [397]	Formal verification Petri Nets	–	–	–

Continued on Next Page...

Table 3.1 – Continued

Author	Testing Category	WS Technology	Tools	Services/ScS Used in Experiments
Li et al. [180]	Model-based testing	WSDL	WS-StarGaze	Parlay X Conference WS, CSTA Routeing Service
Tarhini et al. [313]	Model-based testing	WSDL	–	–
Felderer et al. [106, 107]	Model-based testing	–	–	–
Frantzen et al. [111]	Model-based testing	WSDL	JAMBITON, MINERVA	Synthetic Alarm Dispatcher Service
Yang et al. [367]	Model-based testing	–	–	–
Li and Chou [179]	Model-based testing	–	–	ParalayX Conference service
Liu et al. [189]	Model-based testing	OWL-S	–	–
Maâlej et al. [203, 204]	Model-based testing	BPEL	WSCCT	Synthetic blood search BPEL process
Belli and Linschulte [23]	Model-based testing	BPEL	CPP & ETES	ISELTA web service
Endo et al. [97]	Model-based testing	–	–	Synthetic banking system
Wu and Lee [360]	Model-based testing	WSDL-S	–	–
Sun et al. [304]	Model-based testing	BPEL	–	SmarthShelf synthetic BPEL
Belli et al. [22]	Model-based testing	BPEL	CPP & ETES	ISELTA web service
Offutt and Xu [235]	Fault-based testing	WSDL	–	Mars Robot Communication Service (MRCS)
Xu et al. [364]	Fault-based testing	WSDL	–	MRSC, WS-I example supply chain application
Almedia and Vergilio [79]	Fault-based testing	WSDL	SMAT-WS	9 goverment services
Hanna and Munro [130]	Fault-based testing	WSDL	–	–
Vieira et al. [332]	Fault-based testing	SOAP	–	21 public WS
Tsai et al. [326]	Fault-based testing	OWL-S	–	60 versions of synthetic BBS web service
Martin et al. [207]	Fault-based testing	WSDL	WebSob	–
Salva and Rabhi [280]	Fault-based testing	WSDL	–	Amazon E-Commerce service
Li et al. [181]	Fault-based testing	WSDL	–	5 undisclosed public services
Shafin et al. [287]	Fault-based testing	OWL-S	FIT	Online Bank service from WebLogic
Looker et al. [196, 195]	Fault-based testing	SOAP	WS-FIT	Synthetic stock market trading system with 3 WS
Farj et al. [104]	Fault-based testing	SOAP	NetFIS	Bioinformatics Web services
Juszczyk and Dustdar [158, 159, 157]	Fault-based testing	–	Genesis2	Apache CXF, Groovy SOAP Module, DAIOS

Continued on Next Page…

Table 3.1 – Continued

Author	Testing Category	WS Technology	Tools	Services/ScS Used in Experiments
Siblini and Mansour [291]	Fault-based testing	WSDL	–	Credit card checking service
Mei and Zhang [212]	Fault-based testing	WSDL	–	2 synthetic services (Tritype and Middle)
Lee et al. [174]	Fault-based testing	OWL-S	–	Book finder example OWL-S
Wang and Huang [336]	Fault-based testing	OWL-S	–	Synthetic banking system
Domínguez-Jiménez et al. [86]	Fault-based testing	BPEL	–	Synthetic loan approval
Boonyakulsrirung and Suwannasart [40]	Fault-based testing	BPEL	–	–
Fu et al. [112]	Fault-based testing	–	–	4 Java web service applications (FTPD, JNFS, Haboob, Muffin)
Bessayah et al. [31]	Fault-based testing	SOAP	WSInject	Synthetic TRS
Apilli [7]	Fault-based testing	–	–	–
Watkins [340]	Fault-based testing	–	–	–
Carrozza et al. [56]	Fault-based testing	WSDL	WSRTesting	TerraService, Air Traffic Control services
Oliveira et al. [237]	Fault-based testing	WSDL	–	250 public web services
Bertolino and Polini [30]	Interoperability testing	WSDL	–	–
Yu et al. [377]	Interoperability testing	OWL-S	–	–
Ramsokul and Sowmya [265]	Model-checking for interoperability	WS protocols	–	WS-AT
Guermouche and Godart [126]	Model-checking for interoperability	–	UPPAAL	Synthetic E-government service
Smythe [294]	Interoperability testing	–	–	–
Betin-Can and Bultan [32]	Model-checking for interoperability	–	–	Synthetic travel agency service and purchase order service
Narita et al. [227]	Interoperability testing	WS protocols	WS -VS	An open source WS-Reliability implementation
Andrés et al. [5]	Interoperability testing	–	–	–
Morales et al. [222]	Interoperability testing	–	–	–
Cao et al. [53]	Interoperability testing	–	–	–
Cao et al. [50]	Interoperability testing	–	WSOTF, RV4WS	Product retriever service from Netbeans IDE
Yu et al. [375]	Interoperability testing	–	–	–
Tsai et al.'s [325]	Integration testing	WSDL	Coyote	–

Continued on Next Page…

Table 3.1 – Continued

Author	Testing Category	WS Technology	Tools	Services/ScS Used in Experiments
Huang et al. [142]	Integration testing	–	–	HR management system
Peyton et al. [257]	Integration testing	SOAP,WSDL	TTCN-3	–
Mei et al. [214]	Integration testing	BPEL	–	8 BPEL example applications
Garriga et al. [118]	Integration testing	WSDL	–	Online messenger service
Miao and Liu [220]	Integration testing	WSDL	–	Product management service
De Angelis et al. [80]	Integration testing	BPEL	–	1 synthetic BPEL application
Sun et al. [307]	Integration testing	BPEL	–	Synthetic BPEL examples SupplyChain and SmartShelf
Yu et al. [376]	Interaction testing	OWL-S	–	–
Tsai et al. [323]	Collaborative testing	UDDI	–	–
Bai et al. [18]	Collaborative testing	–	–	–
Zhu [388, 395, 396]	Collaborative testing	–	–	–
Bartolini et al. [19]	Collaborative testing	–	PICASSO	–
Eler et al. [96]	Collaborative testing	–	PICASSO	–
El Ioini and Sillitti [94, 95]	Collaborative testing	WSDL	TestGen4J, SOAPUI	Synthetic bookshop service
Chandrasekaran et al. [68]	QoS Testing	SCET	–	–
Di Penta et al. [83]	QoS Testing	–	–	synthetic audio processing and an existing image manipulation service.
Di Penta et al.'s [82]	QoS Testing	–	–	5 WS from 5 versions of dnsjava
Gu and Ge [124]	QoS Testing	–	–	Synthetic service composition
Palacios et al. [245]	QoS Testing	WS-Agreement	–	–
Bertolino et al. [28]	QoS Testing	–	Puppet	–
Grundy et al.'s [123]	QoS Testing	BPMN, ViTaBal-WS	MaramaMTE	–
Driss et al. [89]	QoS Testing	–	NS-2 simulator	Travel planner
Pretre et al. [261]	QoS Testing	–	iTac-QoS	–
Yeom et al. [370]	QoS Testing	–	–	–
Ruth and Tu [276]	Regression testing	–	–	–
Ruth et al. [274]	Regression testing	–	–	–
Lin et al. [188]	Regression testing	–	–	Purchase order WS
Liu et al. [191]	Regression testing	BPEL	–	–

Continued on Next Page…

Table 3.1 – Continued

Author	**Testing Category**	**WS Technology**	**Tools**	**Services/ScS Used in Experiments**
Mei et al. [213]	Regression testing	BPEL	–	6 BPEL examples from IBM repositiory: atm, buybook, gymlocker, loan approval, marketplace, purchase, triphandling
Srikanth and Cohen [301]	Regression testing	–	–	Enterprise-level SaaS application
Tarhini et al. [311]	Regression testing	–	–	–
Wang et al. [335]	Regression testing	BPEL	–	Example online shop from Tools4BPEL
Mei et al. [216]	Regression testing	BPEL	–	6 BPEL examples from IBM repositiory: atm, buybook, gymlocker, loan approval, marketplace, purchase, triphandling
Li et al. [177]	Regression testing	BPEL	–	Loan approval
Zhang et al. [385]	Regression testing	–	–	Synthetic banking service
Tsai et al. [327]	Test case selection	WSDL, OWL-S	–	–
Mei et al. [213, 217]	Test case prioritisation	BPEL	–	atm, buybook, gymlocker, loan approval, marketplace, purchase, triphandling, buybook and dslservice
Askarunisa et al.[12]	Test case prioritisation	OWL-S	–	two existing web services: weather service and Bible service
Athira and Samuel [13]	Test case prioritisation	–	–	Airline reservation example
Chen et al. [71]	Test case prioritisation	BPEL	–	ATM example
Zhai et al. [383]	Test case prioritisation	–	–	City guide service
Nguyen et al. [231]	Test case prioritisation	WSDL	–	eBayfinder
Nguyen et al. [229]	Test case prioritisation	WSDL	–	eBayfinder
Pautasso [254]	Regression testing	–	JOpera	–
Di Penta et al. [82]	Regression testing	–	–	5 WS from 5 versions of dnsjava
Hou et al. [139]	Regression testing	–	–	Synthetic travel planner system

3.3 Emerging Trends in Service-centric Systems and Testing

In order to speculate upon the future of ScST, the future of SOA and services needs to be discussed first. 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 [149, 273]. Existing sites such as Amazon, Google and Microsoft have transformed their businesses towards services. We believe that this transformation trend will continue to grow faster, especially with the increase in mobile applications and cloud services. According to Flurry [109], a mobile analytics company, it is estimated that mobile application downloads reached 2.6 billion in October 2011. The estimated number of downloads for the same month in 2010 was 600 million. The examples of Bing mobile [35] and Google search [121] application (which are built around services) are a testimony to this transformation.

Traditional Web services fail to exploit full potential of SOA because of difficulties in meeting web service requirements [34]. 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 Semantic Web Services (SWS) will become the new standard service practice [110]. Even though there are many promising initiatives for SWS, unfortunately none of these initiatives have yet been accepted as a standard.

We believe one of the future trends will be testing and certification of web services and ScS. As mentioned by several authors [18, 19, 323], 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 [9] store acts as a provider for many developers while maintaining QoS for the products in the store.

It is also important to identify the open issues in ScST in order to draw a roadmap for the future trends in ScST. As mentioned, the open issues and the issues that require more research in ScST are:

1. Lack of real-world case-studies.
2. Solutions that can generate realistic test data.
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.
6. Modelling and validation of fully decentralised ScS.

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 mentioned in Chapter 2.2, 62% of the publications surveyed provide no experimental results. Furthermore, 22% of the papers, though they did provide experimental results, drew these results from synthetic services or compositions. Only 16% of the papers used real-world case-studies (as depicted in Figure 3.4).

The realistic test data generation problem, mentioned in Section 2.3, is also a major problem in ScST. The importance of realistic test data in testing (especially in ScST) is discussed by Bozkurt and Harman [42] and in Chapter 2. 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 specialised data that existing automated test data generation techniques cannot effectively generate. In such a situation, the tester has two 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 2.9. 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 realistic test data are proposed by Conroy et al. [74] and Bozkurt and Harman [42] (presented in Chapter 4). 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 [186, 205, 208] 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 Zhu et al. [395] 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 though 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 these categories is the use of contracts that provide information such as pre or post-conditions. The second one is the use of models or new stakeholders that provide coverage information to the tester. As mentioned in Section 2.3, 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 Bartolini et al.'s [19] and Eler et al.'s [96] approaches in which coverage information is provided with the involvement of another stakeholder. Salva and Rabhi [279] discuss 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 [5, 25, 222] provide run-time fault detection and a degree of fault localisation using artefacts such as invariants and contracts. Unfortunately, the proposed monitoring approaches primarily check service interactions for prescribed faults.

Decentralised 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. [45] also state the problem of formalising choreographies into standard models and testing them. Recent work [215, 302, 303, 354] address the issues of testing of service choreographies. The number of related publications in this subject compared to the number of publications in testing service compositions shows the need for more research.

3.4 Conclusion

This chapter provided the necessary information that helps build a roadmap into the future of ScST for researchers. The chapter presented the analysis of the current trends in ScST explaining the reasons behind some of the open issues. The overview of all approaches provided in the chapter aimed at highlighting areas where researchers focus and where the gaps are. Lastly, the chapter provided a discussion on the emerging trends in ScST and current open problems that needs more research.

In the following chapters we focus on two of the open problems discussed in this section: automated realistic test data generation and cost reduction. Their importance in ScST is the reason that led us to focus on these subjects.

The realism of test data is not only important in ScST but also important in testing of traditional systems as discussed by McMinn [211]. This problem is becoming increasingly important with the adoption of online APIs, mashups and services. In Chapter 4, we discuss the importance of realistic test data in ScST and introduce the concept of service-centric test data generation as a solution to the realistic test data generation problem.

The problem of cost occurs at different levels of testing, such as human-oracle cost and generation cost in test data generation level, as well as service invocation cost during runtime testing level. For example, in the case of cost of test data generation, the cost might be high if test data is generated manually. This is because manual generation of test data is laborious and error prone [134]. This problem might become more severe for systems where testing frequency is high such as ScS. In order to avoid these costs, test data generation needs to be automated. However, the automation of realistic test data generation is not as straightforward as the others. The current state-of-the art automated approaches fail to generate realistic data effectively and requires the tester to verify the generated inputs. Thus, these approaches incur another type of cost called 'human-oracle cost'. Service-centric test data generation, discussed in Chapter 4, also provides a solution to the automation problem while minimising the human-oracle cost. The cost reduction problem at test data generation level is addressed in Chapter 5 and at runtime level in Chapter 6.

Chapter 4

Automated Realistic Test Input Generation

4.1 Motivaton

One of the most expensive activities in a software project is the construction of test data [147]. As revealed by Chapter 2 and 3, this problem might be more severe in environments like SOA, where testing frequency is high, binding times can be late and services composed of systems provided by many different third parties. One way of reducing this cost is automation of the test data generation process.

Unfortunately, automated test data generation for most online systems is not as straightforward as it is for traditional systems. The problems in test data generation for online systems are mainly caused by the type of required test data. Most online systems require data that cannot be effectively generated by traditional automated test data generation algorithms. For example, the existing semantic web service (SWS) examples from Mindswap [221] require realistic test data, such as ISBNs, United States Zone Improvement Plan (ZIP) codes and English and French words. It is a considerable challenge for any automated test data generation technique to generate realistic test data, not merely 'structurally valid' test data. Realistic data must not only be correctly formed, according to structural rules (which can be automated), it must also have semantics that ties it to real-world entities. This latter property poses challenges to automated testing. More precisely, realistic test data, in this context, is realistic in two respects:

1. Structural Validity: Realistic data must conform to syntactic constraints. For example, a structurally valid ISBN consists of 10 digits $x_1, ..., x_{10}$, such that:

   $$x_{10} = 11 - (10x_1 + 9x_2 + ... + 2x_9)\ mod\ 10.$$

   (This algorithm will be referred as check-digit algorithm in the rest of the present chapter.)

2. Semantic validity: Realistic test data must represent a real-world entity. For example, a realistic ISBN must correspond to a real-world book, not merely a structurally valid ISBN.

Traditional automated testing can generate tests that achieve coverage, while manual testing can generate realistic test cases. The former is efficient yet unrealistic, while the latter is laborious yet realistic. What is needed is a technique that is both realistic *and* automated.

In this chapter, we introduce a novel service-centric test data generation approach that addresses this problem of effectiveness in automated generation of realistic test data. In our approach, the need for previous data and manual tester input are minimised by leveraging the data that can be acquired from compositions of many existing web services.

The advantages of the proposed approach are:

1. **Automated**: Improved effectiveness in automated test data generation for the input types that cannot be effectively generated.

2. **Tailored**: Test data generation/selection based on the tester criteria and optimised test data generation based on the data source.

3. **Applicability**: Ability to generate test data to test any system with semantic information.

4. **Minimised**: Minimal dependence on the existing data sources such as databases and session data.

We named our tool ATAM by combining the initial letters of these four advantages.

The rest of this chapter is organised as follows. Section 4.2 discusses the features and importance of test data in software testing. Section 4.3 introduces our service-centric test data generation approach. Section 4.4 introduces ATAM and the details of tailored data generation process. Sections 4.5 presents our case studies, research questions and our method of investigation. Section 4.6 presents the results from our experiments and answers the research questions. Section 4.7 summarises existing research on test data generation for web services and approaches similar to the proposed approach in other testing application domains. Section 4.8 presents future work and concludes the chapter.

4.2 Features and Importance of Realistic Test Data

The concept of realistic test data and its importance needs to be established clearly before discussing the proposed solution to avoid confusion. Event though, we provided our description of realistic test data and its features in the previous section, we believe we need to discuss further to clear misconceptions.

4.2.1 Features of Realistic Test Data

In order to make our definition clear and explain the features of realistic test data, we will focus on what our definition of realistic test data does not entail.

1) **Realistic test data relates to the requirements of the input and it's free from business requirements of the SUT**. This feature might not be clearly understood from our definition of realistic test data and cause misunderstandings in some situations. For example, this feature can be easily explained for inputs such as ZIP Code or ISBN. However, in generating test data for services such as a compiler service or virtual machine service this feature might cause misunderstandings. In testing the latter services, the tester might assume that a realistic test data (for testing these services) is a source

code that can be compiled or a binary code that can be executed. However, this is not implied in our definition since a realistic test data for these services is a correctly formed file address which points to an existing file. An input such as this is considered a realistic input according to our description.

2) **Realistic test data is not a guaranteed positive or a negative test input**. For example, in the case of an ISBN if it belongs to a book that ISBN number is considered realistic. Even though, it is realistic we cannot expect all web services that requires an ISBN to recognise this ISBN. Thus for some services a realistic test ISBN might not be a positive input. In the case of a compiler service, a file address that points to a file is a realistic input. The correct or incorrect compiling of the file does not affect the realism of this input. For example, if a file that contains Java code might be a positive input for a Java compiler service but for a C# compiler service it might not compile and considered negative input.

There are also exceptions to this feature due to the nature of realistic test data. For any system that validates a given input a realistic test input should always be a positive input. For example, for an ISBN validation service a realistic ISBN is always expected to generate a valid response.

1) **Realistic test data does follow the features of the datatype**. For example, if a certain datatype is temporal, a realistic data of this type is also temporal. Based on the example we discussed earlier, in the case of a test data such as flight location, the realistic test data generated at a certain time will be invalid after a period of time.

4.2.2 Importance of Realistic Test Data

After establishing the concept of realistic test data, we need to explain the importance of realistic test input in testing (especially in ScST). Realism is very important in testing for at least the following four reasons:

1) **Faults Found by Realistic Test Cases are More Likely to be Prioritised for Fixing**: A test case that reveals a fault but uses special values that seldom or never occur in practice is unlikely to attract the attention of a tester who is preoccupied with many

other such 'bug reports' from the field. Finding a test input that convinces the tester of a problem is important. If an automated tool generates many unusual and peculiar inputs that a user is unlikely to use, then the tester will soon abandon the tool or disregard its findings.

2) **Realistic Test Inputs May Denote Important 'Corner Cases'**: There is a role to be played in searching for corner cases that are, in some sense, atypical (and therefore *less* realistic). Such cases as invalid data values are important to test the robustness of the system under test. However, realistic test cases can often denote a kind of 'corner case' too. Suppose there is an input that requires a domain specific value, selected from a narrow potential range of values that lie within a wider input type. For instance, a ZIP code consists of a five digit number, so there are potentially 10^5 correctly typed candidates. However, only very few of these candidate five-digit numbers will correspond to a location in Huntsville, Alabama (these are the 15 codes from 35801 to 35816). Suppose a large part of the system under test is concerned with this particular geographic region and there is a predicate that checks, early on in any user session, whether the address is valid. All code beyond such a predicate test will be inaccessible to (and therefore untested by) any automated testing approach that is unable to focus on the generation of valid Huntsville ZIP codes. We believe that these situations are far from atypical, particularly in ScS, where the services provided carry with them a great deal of inherent domain-specific properties. By generating realistic test data, we are simultaneously providing a mechanism for addressing the problem of generating domain specific data.

3) **Realism is Important for Testing Service Compositions**: When two services are composed, the interface through which they communicate is likely to specialise the kinds of values that may pass between the services. For instance, a generic train booking service might be composed with a service offering a specific holiday package. Suppose that the holiday package may offer only a very limited set of possible train bookings. This restriction creates another kind of 'realistic value'. That is, though the train service must be tested for any possible seat on any train, on any line, at any time

and date, the composition that provides the holiday package requires a specific set of seats on certain trains at restricted times. In generating test cases for such a service composition, a tool that is able to generate realistic test cases for this scenario will only generate correctly restricted journeys. For the same number of test cases, such a tool will be able to test the many of the various aspects of this restricted functionality intensively, while a more general tool will simply repeatedly test the ‘invalid journey’ functionality.

4) **Realistic Test Data Reduces Human Oracle Cost**: McMinn et al. [211] highlighted another benefit of realistic test data: the reduction human oracle cost. Oracle costs may be increased by unrealistic test data that are hard to comprehend and check. This process is laborious and the time required for checking and verifying test data can be undesirably long while generating large test suits.

The need for human oracle might have a bigger impact in SOA testing. This is caused by the fact that in SOA, services need to be tested not only by the developer but also by other stakeholders (such as the third-party certifier and the provider). If the tester does not have adequate domain knowledge to judge whether the test data used is realistic, the test results using this data might not provide the expected level of assurance.

4.3 Service-centric Test Data Generation

In this section, we introduce the concept of service-centric test data generation, explaining every step of the data generation process from service discovery to input generation.

4.3.1 Data Categorisation

In our approach, each input type lies in one of three categories:

1. *Data Originated from a Service (DOS)* is the type of data that can be attained from an existing web service.

2. *Tester Specified Data (TSD)* is the type of data that needs to be provided by the tester (such as a login and password). When our tool requires an input data that

cannot be generated, it requests assistance from the tester.

3. *Method Independent Data (MID)* is the type of data which can be systematically generated. In order to be identified as MID, a data generation method for this input type must be known to ATAM prior to the test data generation process.

MID data can be classified into two groups based on the structural validity criterion:

> **Formed data** is a type of data where a random but structurally valid instance of the data can be generated.

An example of this category is an ISBN generated using the check-digit algorithm.

> **Non-formed data** is randomly generated data that lies within a supertype for which even structural validity cannot be guaranteed.

An example of this category is an 'ISBN' constructed merely by generating 10 random digits; the set of all 10 digit numbers is a supertype of the ISBN type.

All non-MID data is initially considered to be TSD data. However, for TSD data that can be provided by a web service composition becomes DOS data within our test data generation process. Our goal is to maximally automate the production of realistic test data by migrating TSD to DOS.

4.3.2 Services as Data Sources

To address this issue of realism in automated test data generation of services, we propose to use services themselves as the solution to the problem. We shall seek compositions of services that are able to build realistic test cases. If Service A is not able to generate the required realistic value then we will seek to compose it with Service B, the output of which, when fed to service A does result in the desired set of possible realistic values. For example, if we have a service that can produce an ISBN given an author and a service that can generate and author name given a keyword, then the composition can generate an ISBN from a keyword. When further composed with a dictionary service, we obtain a service composition that can produce a valid ISBN with little or no input. In this way, we seek arbitrary service compositions that will progressively build the realism we seek into the test data we seek.

Our approach is limited to semantic systems. The reason for this limitation drives from the need for semantic information concerning input and output messages, which are core elements of our approach. Semantic information for messages not only allows us to automatically discover existing web services with required data, but also allows automatically discover relations among different data structures that might contain the required test data.

ATAM is intended to be used with all semantic systems. As a result, it uses two different types of initial user inputs. If the service/system under test (SUT) is an SWS the tester can provide a semantic service description such as OWL-S description. For any other system, the tester has to provide a semantic description of the required input data in the form of an ontology class. The overall flow graph of our search, analyse and compare process implemented by ATAM, is depicted in Figure 4.1.

The information on test data to be generated is passed to the ontology analyser to search for possible relations defined within the specified ontologies.

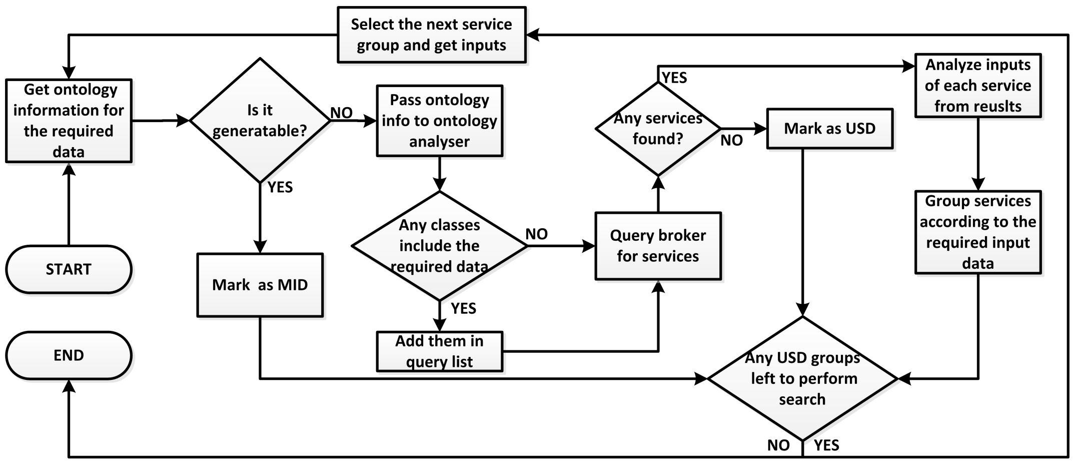

Select the next service group and get inputs
Get ontology information for the required data
Is it generatable?
NO
YES
Pass ontology info to ontology analyser
Any classes include the required data
NO
YES
Add them in query list
Query broker for services
Any services found?
YES
NO
Mark as USD
Analyze inputs of each service from reuslts
Group services according to the required input data
START
END
Mark as MID
Any USD groups left to perform search
NO
YES

Figure 4.1: Flow graph of the overall search process that seeks realistic test data from composition of other services

4.3.3 Ontology Analyser

Automated service discovery is one of the main topics of SOA. In our approach, we benefit from the ability provided by SOA to automatically discover services that provide the required test data.

There are several existing tools that provide subsumption based service discovery/-matchmaking. Subsumption relation-based matchmaking is capable of finding super or subclass of the provided ontology class. For example, suppose we take an ontology where *book class* is defined as a subclass of *ISBN class* and try to discover services using this ontology. In this scenario, it is possible to discover the services (among the web services using this ontology) that provide book information while searching for services that provide ISBN.

Discovering the relations between service messages, such as ISBN and book, is one of the most important requirements that needs to be facilitated in our approach. Unfortunately, this direct relation might not be specified in the ontology. We analysed many of the available ontologies that include a definition of ISBN in Swoogle [308], a popular ontology database, and found that none of the existing ontologies with an ISBN definition contains this relation. Interestingly, in all except one ontology, 'ISBN' is defined as a *DatatypeProperty*, a property used in defining relation between instances of an ontology class and RDF literals and XML Schema datatypes, of book class.

In light of the results from our analysis, we introduced an ontology analyser that discovers relations between ontology entities. The ontology analyser currently discovers two different possible relations among classes and properties. The discovered relations are, as mentioned earlier, the relation between a DatatypeProperty and its defining classes, and relation between two separate classes defined using a Datatype-Property. The ontology analyser is not only able to process a single ontology, but it is also able to process ontologies where relations to external classes are specified as well.

4.3.4 Service Discovery and Search Result Processing

After the ontology analysis stage, ATAM queries the service broker(s) for services that provide any of the discovered ontology classes as output. Search queries used in the experiments include only an output specification.

In order to achieve this functionality, the generated search queries use the Ontology Web Language (OWL) ‘Thing’ class, a built-in class ‘global base class’ that denotes superclass of all classes. The reason for using ‘Thing’ class is to find all available web services with the given output message. Using this class as input with a subclass based subsumption criteria on inputs guarantees a service matching regardless of the service inputs.

After obtaining the search results, services from the results need to be analysed for their inputs. This is to identify whether they require MID or TSD type input data, as explained in Section 4.3.1. After the analysis, services with TSD type inputs go trough an additional iteration of the discovery process in order to investigate whether there exist services that provide inputs of this service.

As mentioned in Section 4.3.1, some of the TSD can be provided by other web services. A TSD service with an input that can be provided by another service becomes a DOS service. When a web service proving the data for another TSD service is found, these two services are considered to form an abstract service composition (referred to as composition). In compositions, the service providing the data is considered as higher level than the service that requires the data.

A search result might contain many TSD services with the same functionality. In such a situation, performing the discovery process for all TSD services involves running the same query several times. This is an unwanted side effect that increases the time and cost of service discovery.

In order to avoid this potential overhead, a grouping stage is introduced. During this stage, web services from search results are grouped based on their inputs. Services with same input(s) are grouped together and a single service from each category is used to represent the group. As a result, all the services in a group require a single query for each input data they need.

During the grouping process, services also go through an elimination process. At present, we introduced the following two natural elimination criteria:

1. Services that require the same input data as SUT are eliminated.
2. Services that already exist in lower levels in the existing compositions are elimi-

nated.

4.3.5 Services with multiple inputs

Using services that require multiple inputs is more expensive than those that require a single input. Of course, it may not be possible to avoid services with multiple inputs, so we also considered ways to reduce the cost of using these services.

One obvious solution is to reduce the number of services that provide the required inputs. There might be situations where several of the required inputs of a service can be provided by a single web service. In such situations, it is possible to reduce the execution time of the test data generation process by reducing the number of search queries and service invocations. In order to achieve this goal, a 'partition stage' is introduced.

In this stage, the ontology analyser finds classes that contain the required data for each input as normal. After the initial analysis, the ontology analyser examines the resulting classes for the possibility of containing an input other than that for which the search was conducted. If an ontology class containing more than one TSD input is found, it is marked as a combined solution for all the inputs it contains. As a result, during the next search iteration only one search query is used on combined solutions rather than the number of inputs they provide.

This functionality is only partly developed in the present version of ATAM. Its extension and other efficiency-improvement technique remain topics for future work.

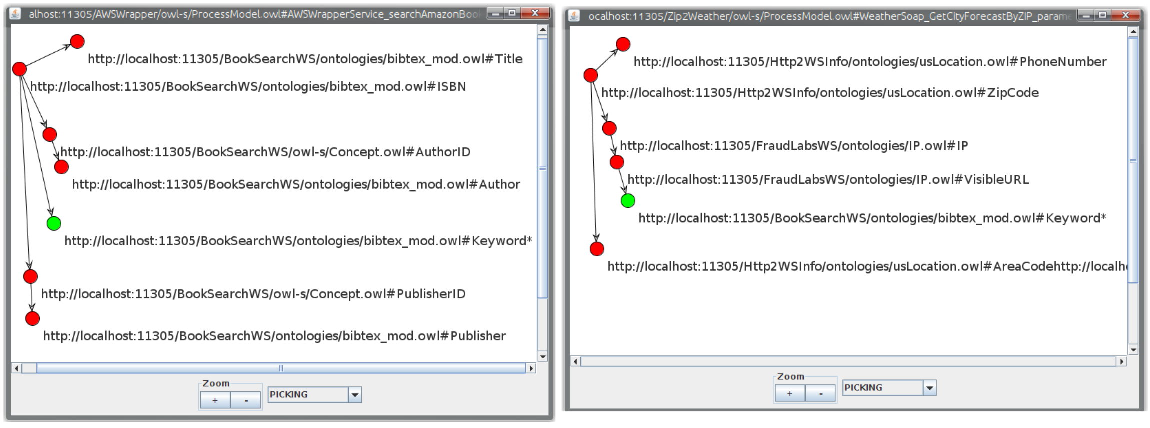

alhost:11305/AWSWrapper/owl-s/ProcessModel.owl#AWSWrapperService_searchAmazonBoo
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Title
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#ISBN
http://localhost:11305/BookSearchWS/owl-s/Concept.owl#AuthorID
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Author
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Keyword*
http://localhost:11305/BookSearchWS/owl-s/Concept.owl#PublisherID
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Publisher
Zoom
+
-
PICKING
ocalhost:11305/Zip2Weather/owl-s/ProcessModel.owl#WeatherSoap_GetCityForecastByZIP_parame
http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#PhoneNumber
http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#ZipCode
http://localhost:11305/FraudLabsWS/ontologies/IP.owl#IP
http://localhost:11305/FraudLabsWS/ontologies/IP.owl#VisibleURL
http://localhost:11305/BookSearchWS/ontologies/bibtex_mod.owl#Keyword*
http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#AreaCodehttp://localh
Zoom
+
-
PICKING

Figure 4.2: Search results for CS1 and CS2 respectively from left to right

4.4 ATAM and Tailored Test Data Generation

In this section we introduce ATAM and how it enables tailored data generation. Details of error handling mechanisms implemented in ATAM are also presented in this section.

4.4.1 Tailored Test Data Generation

After the completion of the search process, ATAM presents the tester with a graphical view of the discovered compositions in an interactive GUI as shown in Figure 4.2. This window provides the tester with the group view of the compositions and it also allows the tester to see the details of each web service in each group.

In the GUI, nodes with green colour represent MID service groups which can be automatically generated by ATAM. The red nodes represent TSD and DOS services. The node that all compositions are connected to represents the searched data.

The tester starts the data generation process from the node at the end of one composition. This might be the case for compositions that end with a MID group. On the other hand, for compositions that end with a TSD service, the tester might want to start the process from a different group. ATAM allows the tester to start the test data generation process at any point in a composition.

If the tester decides to start from a TSD group, ATAM asks the tester to provide the input data for the starting group in order to invoke a service from the selected group. Similarly, if the tester selects a DOS group then this group is considered to be a TSD group and the tester is asked for inputs.

This feature handles those cases where no MID services are found. It provides a ‘basecase’, allowing the tester to start test data generation process from a service group where he/she has knowledge on the required input data.

4.4.2 Constraints for Tailored Test Data Generation

When the tester selects a service group to start, ATAM asks the tester for constraints. Constraints are not essential but they do allow tailoring of test data. Using constraints, the tester can refine the outputs of each service group, thereby guiding the test data generation process towards the requirements of the SUT. The constraint input window and all windows a tester can provide inputs to ATAM is depicted in Figure 4.3.

To eliminate outputs that do not satisfy the constraints, we use XPath queries in ontology class instances. These queries are provided by the service developer within the *xsltTransformationString* element of the OWL-S grounding. As a result, the tester is not expected to provide a complete query, merely the constraint part of the XPath query such as "contains(hasAuthor,'rowling')" or "hasPublisher = 'Springer'".

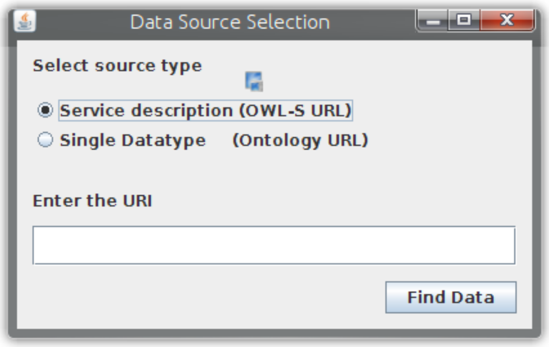
The ontology analyser helps the tester to build constraints by providing the structures of ontologies. The tester is provided with this information through a constraint input window.

4.4.3 Service & Data Error Handling

During test data generation process, errors might occur due to service faults or the datum that are passed among services. ATAM handles both types of error. Service-specific errors, such as interface errors and availability problems, are handled by the soapUI libraries. For other errors, such as SOAP faults and web server errors, ATAM introduces an error handling mechanism. Error messages from soapUI libraries are fed into this mechanism. In the case of a an error, ATAM replaces the erroneous service with another service from the same group. Where such an alternative exists, this replacement process happens seamlessly without interrupting the test data generation process. ATAM records the input data that are used for each service group in order to avoid using the same input multiple times. In the case of an error, the corresponding input is discarded.

4.4.4 Implementation Details

The selection of semantic service specifications is based on expedient of tool availability. That is, the OWL-S semantic markup language is chosen due to the availability of tools that support OWL-S. WSDL2OWLS [358] is used to automatically generate OWL-S templates from WSDL documents. JXML2OWL [160] is used to generate the transformation strings in the OWL-S grounding. WSDL2OWLS uses OWL-S version 1.1 specifications and generates OWL-S specifications. The architecture of ATAM based on the selected service specifications is depicted in Figure 4.4.



Data Source Selection

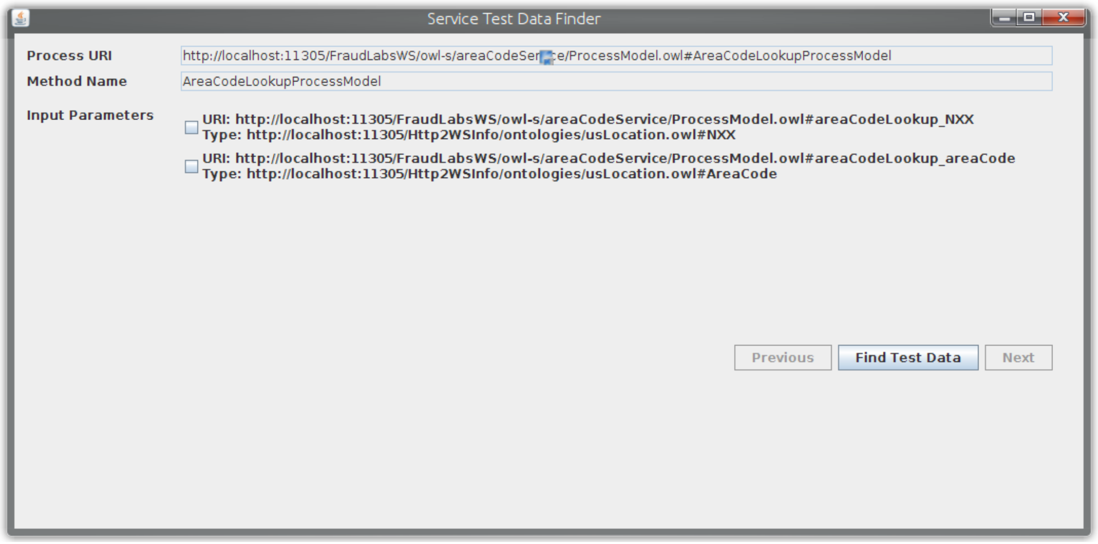
Select source type

[✓ Service description (OWL-S URL)]

Single Datatype (Ontology URL)

Enter the URI

(a) Ontology input type selection window



Service Test Data Finder

Process URI: http://localhost:11305/FraudLabsWS/owl-s/areaCodeService/ProcessModel.owl#AreaCodeLookupProcessModel

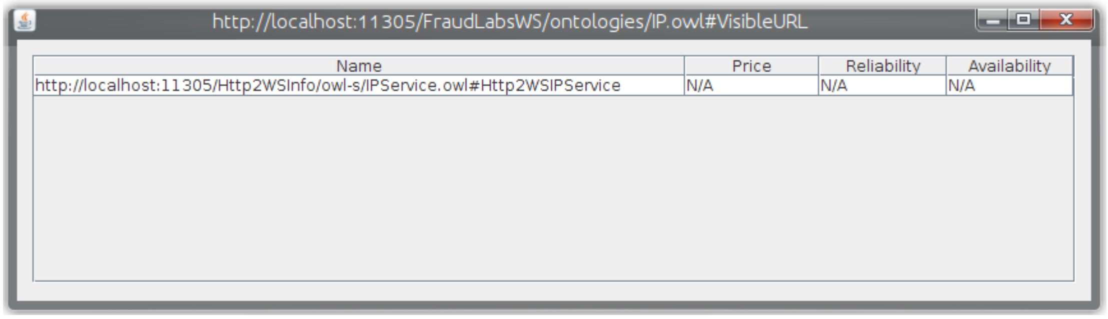
Method Name: AreaCodeLookupProcessModel

Input Parameters:

URI: http://localhost:11305/FraudLabsWS/owl-s/areaCodeService/ProcessModel.owl#areaCodeLookup_NXX
Type: http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#NXX

URI: http://localhost:11305/FraudLabsWS/owl-s/areaCodeService/ProcessModel.owl#areaCodeLookup_areaCode
Type: http://localhost:11305/Http2WSInfo/ontologies/usLocation.owl#AreaCode

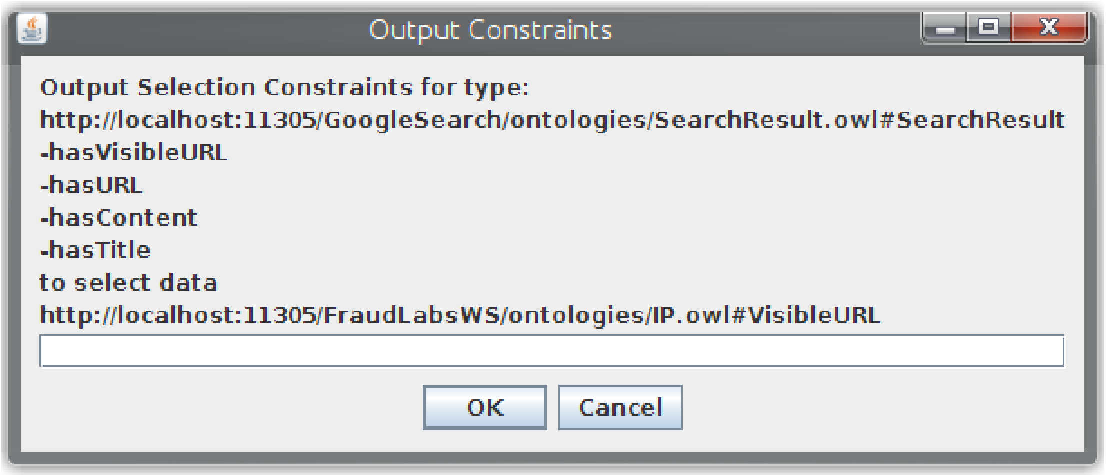
(b) Method and input selection window (for test data generation using OWL-S specification)



http://localhost:11305/FraudLabsWS/ontologies/IP.owl#VisibleURL

Name	Price	Reliability	Availability
http://localhost:11305/Http2WSInfo/owl-s/IPService.owl#Http2WSIPService	N/A	N/A	N/A

(c) Service group information window



Output Constraints

Output Selection Constraints for type:
http://localhost:11305/GoogleSearch/ontologies/SearchResult.owl#SearchResult
-hasVisibleURL
-hasURL
-hasContent
-hasTitle
to select data
http://localhost:11305/FraudLabsWS/ontologies/IP.owl#VisibleURL

(d) Tester constraint input window

Figure 4.3: ATAM user input widows

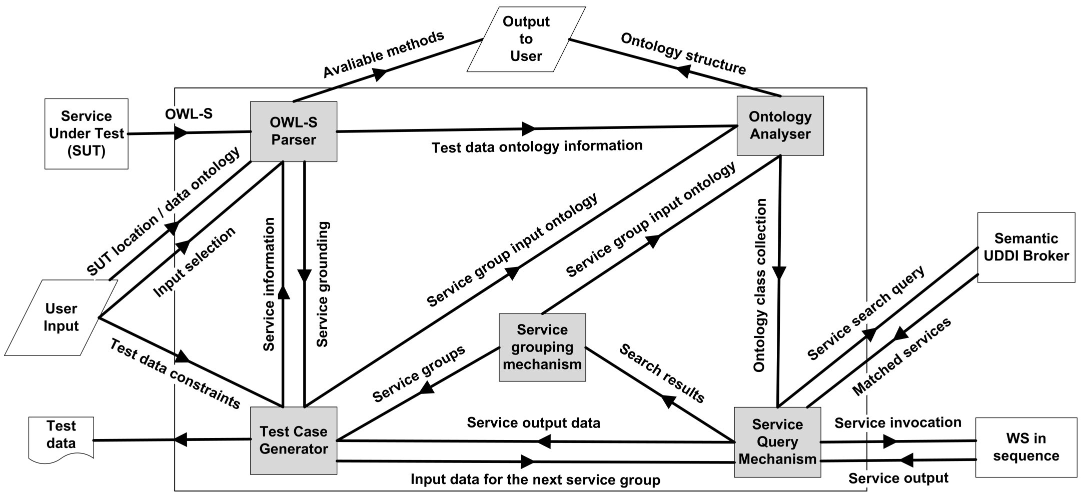


Figure 4.4: Overall architecture of ATAM

Even though all the OWL-S specifications used in our experiments are in OWL-S version 1.1 format, the OWL-S parsing mechanism uses the OWL-S API version 2. This was necessitated by a major technical problem faced with OWL-S API 1.1 which was the API's inability to parse xsltTransformationString elements correctly. Our use of two different OWL-S versions creates issues, such as grounding incompatibilities, which we had to solve by modifying the OWL-S Grounding structure.

ATAM supports OWL Full, a version of OWL that allows mixing of RDF Schemata with OWL, and which enjoys a high degree of compatibility with RDF. Even though ATAM supports OWL Full, it does not support pure RDF ontologies due to constraint incompatibilities. However, despite these technical details and compatibility issues, we have been able to use ATAM to provide realistic test data, as our two case studies illustrate.

The service query mechanism uses the open source version of the soapUI [298] framework to send and receive SOAP messages. The soapUI framework supports multiple messaging protocols for services REST and SOAP. The web service broker/matchmaker selection is based on OWL-S support. The Alive matchmaker [2] was used for simulating a real-world semantic service broker. It is open source and offers flexible search query generation.

4.5 Empirical Studies

In this section, we introduce the case studies we used in our experiments, present the research questions we asked and explain our method of investigation.

4.5.1 Case Studies

For the experiments, we selected the set of related services shown in Table 4.1, most of which are commercial real-world services. We used these services for two reasons:

1. Commercial services are more likely to exhibit issues concerned with 'realistic data'.
2. Commercial services allow a degree of 'real-world' evaluation of our ATAM approach.

	Service	**Input(s)**	**Output**
Case Study 1	ISBNdb by ISBN (ISBNS)	ISBN	Book
	ISBNdb by keyword (ISBNK)	Keyword	Book
	ISBNdb by title (ISBNT)	Title	Book
	ISBNdb find author ID	Author	AuthorID
	ISBNdb by author ID	AuthorID	Book
	ISBNdb find publisher ID	Publisher	PublisherID
	ISBNdb by publisher ID	PublisherID	Book
	Amazon Book Search (ABS)	ISBN	Item
	Validate ISBN (vISBN)	ISBN	Boolean
Case Study 2	Google Search (GS)	Keyword	Search result
	Strikeiron IP Lookup (SIP)	Http address	IP address
	FraudLabs IP2Location (FIPL)	IP address	US location
	CDYNE CheckPhone (CC)	Phone number	US location
	FraudLabs AreaCodeWorld	NPA + NXX	US location
	ZipCode Validator (ZCV)	ZipCode	Boolean
	CDYNE Weather (CW)	ZipCode	W. forecast

Table 4.1: Web services used in the experiments

Only 6 out of 16 these services are free to use: ABS, vISBN, ZCV, GS, CW and CC. Even though these services are free, most of them are provided by commercial companies such as Google, Amazon and CDYNE. We only used the GS service in test data generation. The rest of the free services were used as services under test in the experiments.

One side effect of using commercial services is the potentially prohibitive cost involved in running our experiments. Fortunately, for most of the services used in our case studies, we have been given limited but sufficient access which facilitated our experiments. However, we could not get cost-free access for the SIP service. As a result, we had to emulate the functionality of the SIP service by creating a SOAP service that mimics the SIP service's functionality. This new SOAP service relies on the Selfseo [284] website to provide the necessary data.

Some of the services are wrapped in SOAP web service wrappers in order to overcome two technical problems:

1. Most of these services are REST services. Unfortunately, WSDL2OWLS tool does not support REST services even those with a WSDL description.
2. Some of these services are intended to be used statically. For example, ABS requires a signature with each SOAP message and can not be invoked dynamically.

4.5.2 Research Questions

We ask the following research questions:

RQ1. Is our approach more effective than random test data generation in generating realistic data?

RQ2. Is our approach more effective than random test data generation in situations where formed test data can be generated?

RQ3. Does using formed data rather than non-formed data as MID input data affect the effectiveness of our approach?

RQ4. Can our approach effectively generate tailored test data which fulfils given constraints?

4.5.3 Method of Investigation

In order to answer the research questions, we use two real-world case studies: CS1 and CS2. The services in each case study are listed in Table 4.1.

In CS1, we seek to generate realistic ISBNs using existing services. In order to evaluate the effectiveness of our approach, we selected vISBN and ABS services for evaluation and the rest of the services for generating ISBNs. vISBN is used to verify the structural validity of ISBNs and ABS is used to check if the ISBNs represent a real-world book. The search process for this case is presented in Figure 4.5 and the resulting service compositions are presented in Figure 4.2.

In CS2, we seek to generate realistic ZIP codes. In this case study, we selected ZCV and CW services for evaluation and the rest of the services for generating ZIP codes. ZCV is used to verify the structural validity of the generated ZIP codes and ABS is used to check whether the generated ZIP codes represent a real-world location in US. The resulting compositions for this case are presented in Figure 4.2.

In order to answer RQ1, we generated 100 ISBNs and 100 ZIP codes using both our approach and random test data generation. Then we compared the effectiveness of these approaches in creating structurally valid test cases (valid ISBNs) and in testing real-world services (i.e. generating semantically valid test cases.)

For the answer to RQ2, we generated 100 unique formed ISBNs using the check-digit algorithm. Then we used the same evaluation criteria used for RQ1 and compared ISBNs generated by our approach against the formed ones.

For the answer to RQ3, we introduced a keyword generator that generates valid English words. Using this generator, we generated 100 unique formed inputs and also generated unique 2 character long strings as non-formed data. The generated input data was used as input to service compositions that require *keyword* as input; ISBNK and GS. We ran these inputs and compared their effectiveness at test data generation with our approach.

There are two methods for generating tailored test data: the MID-seed method and the TSD-seed method. In the MID-seed method, ATAM generates test data using MID input data while eliminating data that does not fulfil the constraints. The TSD-seed method uses selected TSD input data which intend to direct ATAM towards data that fulfils the constraints. In order to answer RQ4, we evaluated these two methods using both case studies with five different constraints for each case study.

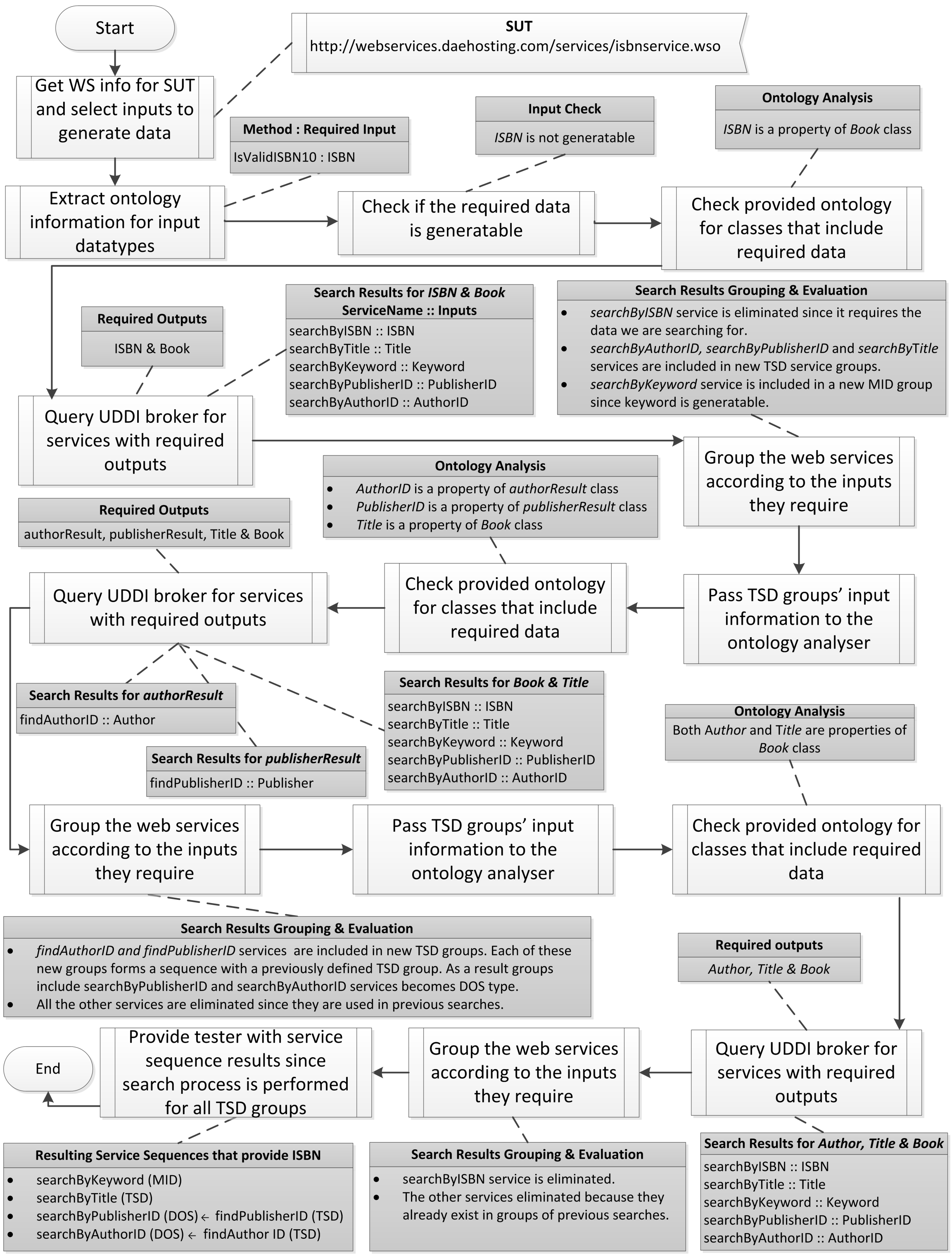

Start
SUT
http://webservices.daehosting.com/services/isbnservice.wso
Get WS info for SUT and select inputs to generate data
Method : Required Input
IsValidISBN10 : ISBN
Input Check
ISBN is not generatable
Ontology Analysis
ISBN is a property of Book class
Extract ontology information for input datatypes
Check if the required data is generatable
Check provided ontology for classes that include required data
Required Outputs
ISBN & Book
Search Results for ISBN & Book
ServiceName :: Inputs
searchByISBN :: ISBN
searchByTitle :: Title
searchByKeyword :: Keyword
searchByPublisherID :: PublisherID
searchByAuthorID :: AuthorID
Search Results Grouping & Evaluation
searchByISBN service is eliminated since it requires the data we are searching for.
searchByAuthorID, searchByPublisherID and searchByTitle services are included in new TSD service groups.
searchByKeyword service is included in a new MID group since keyword is generatable.
Query UDDI broker for services with required outputs
Group the web services according to the inputs they require
Ontology Analysis
AuthorID is a property of authorResult class
PublisherID is a property of publisherResult class
Title is a property of Book class
Required Outputs
authorResult, publisherResult, Title & Book
Query UDDI broker for services with required outputs
Check provided ontology for classes that include required data
Pass TSD groups' input information to the ontology analyser
Search Results for authorResult
findAuthorID :: Author
Search Results for publisherResult
findPublisherID :: Publisher
Search Results for Book & Title
searchByISBN :: ISBN
searchByTitle :: Title
searchByKeyword :: Keyword
searchByPublisherID :: PublisherID
searchByAuthorID :: AuthorID
Ontology Analysis
Both Author and Title are properties of Book class
Group the web services according to the inputs they require
Pass TSD groups' input information to the ontology analyser
Check provided ontology for classes that include required data
Search Results Grouping & Evaluation
findAuthorID and findPublisherID services are included in new TSD groups. Each of these new groups forms a sequence with a previously defined TSD group. As a result groups include searchByPublisherID and searchByAuthorID services becomes DOS type.
All the other services are eliminated since they are used in previous searches.
Required outputs
Author, Title & Book
End
Provide tester with service sequence results since search process is performed for all TSD groups
Group the web services according to the inputs they require
Query UDDI broker for services with required outputs
Resulting Service Sequences that provide ISBN
searchByKeyword (MID)
searchByTitle (TSD)
searchByPublisherID (DOS) ← findPublisherID (TSD)
searchByAuthorID (DOS) ← findAuthor ID (TSD)
Search Results Grouping & Evaluation
searchByISBN service is eliminated.
The other services eliminated because they already exist in groups of previous searches.
Search Results for Author, Title & Book
searchByISBN :: ISBN
searchByTitle :: Title
searchByKeyword :: Keyword
searchByPublisherID :: PublisherID
searchByAuthorID :: AuthorID

Figure 4.5: Complete search process for the first Case Study (CS1), explaining each step of the service composition generation in detail. The search process for CS1 is important because it illustrates the effectiveness of our grouping mechanism in finding the shortest compositions. The whole search process presented in this figure is automated. As a result, ATAM considerably reduces the manual effort required for service-based realistic test data generation.

4.6 Results and Analysis

In this section, we present the results from our experiments, provide answers to the research questions we asked and discuss the threats to the validity of the experiments we conducted.

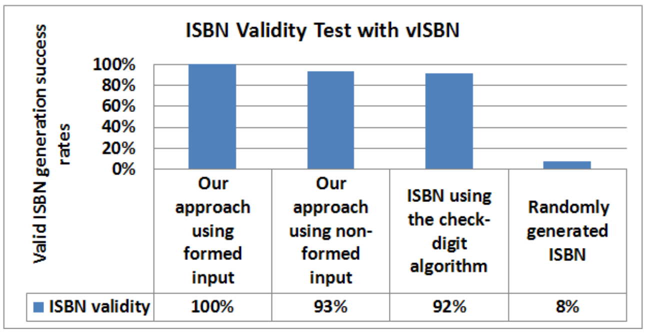
4.6.1 Results

In generating valid ISBNs (depicted in Figure 4.6), our approach achieved a 100% success rate using formed data and 93% using non-formed data. Random test data generation achieves only an 8% success rate in generating valid ISBNs, whereas ISBNs generated using the check-digit algorithm achieved a 92% success rate.

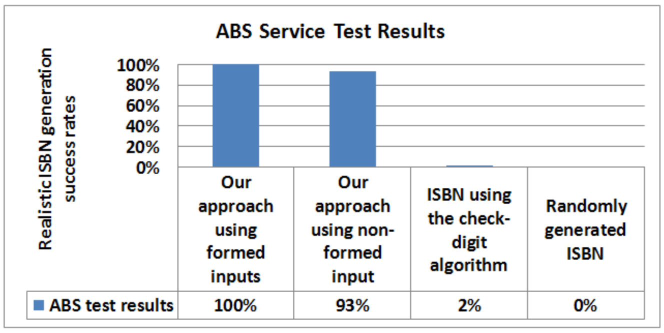
In testing the ABS service (depicted in Figure 4.6), our approach achieved a 100% success rate with formed data and a 93% success rate with non-formed data. Only 2 out of 92 valid ISBNs generated with the ISBN validation algorithm successfully tested the ABS service. Valid ISBNs are unlikely to correspond to real books. As expected, none of the randomly generated ISBNs tested ABS successfully.

There are two unexpected results in Figure 4.6. The first one is the 92% validity score for ISBNs generated using the check-digit algorithm. This result is unexpected because we expected all ISBNs generated using this algorithm to be valid. We tested the 8 invalid ISBNs (4663277421, 2546962401, 8135892901, 5265526961, 7445203631, 1976505021, 3914584191, 6863093121) with other online ISBN validation tools. Unexpectedly, some of these tools identified these ISBNs as valid while others did not. This further confirms the need for realistic test data, even in cases where the syntactic structure is well defined. Even in these apparently straightforward cases, we cannot be sure to have valid instances without real-world validation of test cases.

The second unexpected result is the 93% validity score for our approach with non-formed inputs using ISBNK service. The same inputs achieved 96% validity score when used with ISBNT service. This is unexpected because the ISBNT service returns only a subset of possible ISBNK service results with the same input. According to developer functionality description ISBNK perform searches for books using all details



(a) Comparison of valid ISBN generation success rates



(b) Comparison of realistic ISBN generation success rates

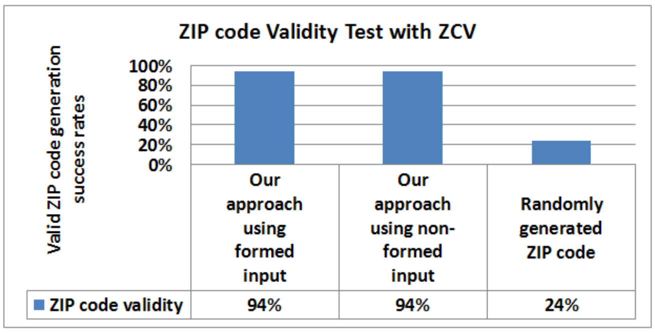
Figure 4.6: Comparison of overall success rates for our approach against random test data generation for the first case study: the image on the top presents the effectiveness at successfully generating structurally valid test data. In this evaluation, our approach achieved a 100% success rate with formed input data and a 93% success rate using non-formed input data. On the other hand, 92% of the ISBNs generated using the check-digit algorithm and only 2% generated using random generation were found to be structurally valid. The image on the bottom presents the effectiveness at generating realistic test data. Our approach achieved 100% success rate with formed input data and a 93% success rate using non-formed input data. Whereas only 2% of the 'ISBNs' were generated by the check-digit algorithm and none of the randomly generated ISBNs successfully tested a real-world service.

of the indexed book information (which includes the title field) and ISBNT perform searches using only the title field.

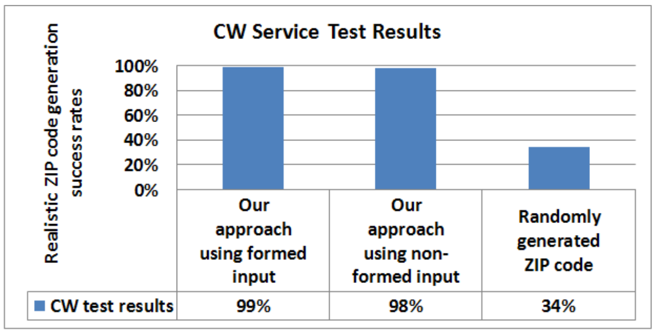
To the best of the authors' knowledge, there is no algorithm to generate structurally valid ZIP codes other than requiring that they consist of 5 digits. As a result, in the second case study (CS2), our approach is compared to random test data generation only. For the problem of structurally valid ZIP code generation (depicted in Figure 4.7), our approach achieved a 94% success rates using both formed and non-formed data, whereas random test data generation only achieved a 24% success rate.

In testing the CW service (depicted in Figure 4.7), our approach achieved a 99%

success rate with formed data and a 98% success rate with non-formed data. On the other hand, only 34% of the randomly generated ZIP codes successfully tested the CW service.



(a) Comparison of valid ZIP code generation success rates



(b) Comparison of realistic ZIP code generation success rates

Figure 4.7: Comparison of overall success rates for our approach against random test data generation for the second case study. The image on the top presents the effectiveness in successfully generating structurally valid test data. In this evaluation, our approach achieved a 94% success rate with both formed and non-formed input data. On the other hand, only 24% of the ZIP codes generated using random generation were found out to be structurally valid. The image on the bottom presents the effectiveness at generating realistic test data (structurally and semantically valid). In generating realistic data, our approach achieved a 99% success rate with formed input data and a 98% success rate using non-formed input data. Whereas, only 34% of the randomly generated ZIP codes successfully tested a real-world service.

The service reliability problem can be observed by comparing the results of CW and ZCV tests with the same ZIP codes. After investigating possible underlying causes of these inconsistencies in results, we discovered the following:

1. CW is not 100% reliable and returned unexpected results for 7 valid ZIP codes (20001, 20005, 20212, 03867, 04469, 20420, 97086). This also is the reason for our approach scoring less than 100% using formed input data.

2. ZCV's reliability appears to be lower than of CW. As a result, the achieved success rates are lower than expected.

3. The website used by the SIP replacement service also exhibited reliability issues. This caused a lower score than expected in 2 cases.

In order to test the ability to generate tailored test data, we used two different scenarios for the case studies. For CS1, we tried to generate an ISBN which is published by a chosen publisher. For CS2, we tried to generate a ZIP code which belongs to a given state. In this part of the experimental analysis, we did not compare our approach to random test data generation due to the expected low success rates it would achieve based on the previous evaluations.

Publisher	**Constraint**	**Success Rate**
Random House	contains(hasPublisher,'Random House')	5%
Pearson	contains(hasPublisher,'Pearson')	2%
Hachette	contains(hasPublisher,'Hachette')	0%
HarperCollins	contains(hasPublisher,'HarperCollins')	7%
Simon&Schuster	contains(hasPublisher,'Simon&Schuster')	3%

Table 4.2: Success rates for tailored ISBN generation using MID-seed

State	**Constraint**	**Success Rate**
California	contentsEqual(State,'CA')	93%
Texas	contentsEqual(State,'TX')	29%
New York	contentsEqual(State,'NY')	17%
Floria	contentsEqual(State,'FL')	7%
Illinois	contentsEqual(State,'IL')	5%

Table 4.3: Success rates for tailored ZIP code generation using MID-seed

As mentioned in Section 4.5.3, there are two different methods for tailored test data generation depending upon whether MID or TSD is used as a 'seed'. Table 4.2 and Table 4.3 present the results for the MID-seed method. By analysing the results, we concluded that in most cases ATAM is able to find a test input that fulfils the given criterion. Unfortunately, the success rates for the MID-seed method vary greatly depending on the selected criterion and the test data domain. The achieved success rates can be as high as 93% (as shown in Table 4.3) and as low as 0% (as shown in Table 4.2).

Publisher	**Used input**	**Success Rate**
Random House	Random House	100%
Pearson	Pearson	100%
Hachette	Hachette	100%
HarperCollins	HarperCollins	100%
Simon & Schuster	Simon & Schuster	100%

Table 4.4: Success rates for tailored ISBN generation using TSD-seed

Table 4.4 and Table 4.5 present the results for the TSD-seed method. Using the TSD-seed method, ATAM successfully generated test data fulfilling the given constraints in both cases studies and achieved a 100% success rate.

State	**Used input**	**Success Rate**
California	website in California	100%
Texas	website in Texas	100%
New York	website in New York	100%
Florida	website in Florida	100%
Illinois	website in Illinois	100%

Table 4.5: Success rates for tailored ZIP code generation using TSD-seed

4.6.2 Answers to Research Questions

The results from the experiments provide evidence for the effectiveness of our approach compared to random test data generation and give an answer to RQ1. In all the scenarios, our tool noticeably outperformed random test data generation. On the other hand, results from both case studies indicated that random generation is not effective when it comes to generating realistic test data. However, existing service testing, whether it is automated at all, relies heavily on random testing. The results from CS1 clearly show how ineffective random generation can be in some domains. In CS1, random generation failed to generate a single ISBN that successfully test ABS service. Even the highest success rate achieved by random generation (34% in CS2) is significantly lower than the success rate of our approach in the same case study.

For RQ2, the results shows that even with a valid generation algorithm random test data generation cannot guarantee a high success rate for realistic test data generation. Even though the ISBNs generated using ISBN validation algorithm achieved a high structural validity rate, as shown in Figure 4.6, these ISBNs achieved only 2% success rate when testing the ABS service. On the other hand our approach for the same scenario achieved 100% success rate with formed input data.

The benefits of using formed input data can be observed in the results of both case studies, as shown in Figure 4.6 and Figure 4.7. Using formed data achieved 100% success rates in both case studies, while non-formed data achieved 93% and 98% success rates. These results provide evidence that using formed input data allows our approach to achieve better testing (answering RQ3).

In terms of tailored test data generation, as asked in RQ4, the results provide evidence for our approach's ability to generate test data fulfilling given constraints. This functionality has the potential to achieve 100% success rates using TSD-seed method (as shown in Table 4.4 and Table 4.5). Unfortunately, the success rates are not as consistent with MID-seed method (as shown in Table 4.2 and Table 4.3).

4.6.3 Threats to Validity

In the experiments, the most important external threat which limits the general applicability of the proposed approach is the test scenarios representing real-world ones. This concern is especially important for work in which experimental validation is performed using simulation and synthetic case studies. In fact, in our experiments there are few synthetically generated artefacts (such as ontologies and service wrappers) which might lead to a sense of unrealism. In order to clarify these issues, we discussed how we maintained realism in all aspects of the experiments below:

1. *Realistic ontologies*: Due to a lack of well-structured ontologies that represent the concepts used in our experiments, we were forced to generate ontologies. As well as the representation problem, there were two main technical problems with existing ontologies: syntactic problems and incompatibilities with the match-

maker. In order to overcome these problems, we generated new ontologies by modifying existing ontologies. By making minor modifications we achieved two goals: maintained realism and increased conformance to OWL [341] specifications.

2. *Realistic services*: As mentioned, we wrapped some of the services in SOAP services wrappers in order to make them compatible with the tools and service specifications and to maintain their conformity to the ontologies. In order to maintain realism, we implemented the wrappers to not modify data syntax or content. Also, for the SIP service, we introduced a wrapper that simulates its functionality using a website due to access restrictions. SIP provides an 'IP Lookup' service which is a common service provided by many websites (even before the introduction of web services). We chose one of the highly rated websites [284] as a substitute to SIP and manually tested for correct functioning.

3. *Realistic matchmaking and service compositions*: Realism in matchmaking is not an issue since we used an existing matchmaking tool with realistic ontologies. Due to data being unmodified by the wrapper services, generated compositions of these services are some of the possible real-world orchestrations of them.

The only internal threat which might have affected the results obtained is the issue of the correctness of the check-digit algorithm (used in generating valid ISBNs). The correctness of the algorithm is verified using an external ISBN verification service as mentioned in the results.

4.7 Related Work

This section is divided into two categories: test data generation for web services and test data generation approaches that reuse existing resources.

4.7.1 Test Data Generation for Web Services

Test case generation for web services is usually based on the web service specification. Traditional web services provide specifications that include abstract information on the available operations and their parameters. Information from these specifications can be used to support test data generation for black-box testing techniques.

Existing WSDL-based test data generation approaches [16, 20, 29, 130, 185, 201, 235, 295] depend on XML Schema datatype information. The datatype information, which defines various constraints for each input type, allows data generation for each simple type. Since it is possible to define complex datatypes, test data generation for these types simply requires decomposition of the complex type into simple types and application of data generation for each simple type. This approach can generate structurally valid test inputs, but may not generate *realistic* test inputs.

Fault-based test data generation tests for prescribed faults in web services. Unlike other testing approaches, in fault-based test data generation, erroneous test data is generated intentionally. Proposed fault-based testing approaches [130, 235, 326, 332, 364, 384] generate test data from communications among services by inserting erroneous data. The focus of fault-based testing is often the generation of tests able to reveal the fault in question. However, there is no guarantee that the tests so-generated will be realistic.

Test data generation from a WSDL definition is limited to generating structurally valid test data, but this cannot be guaranteed to be realistic because WSDL contains no behavioural information about the service under test. This limitation is reduced with the introduction of semantic web services. The use of semantic models such as OWL-S for test data generation is proposed [17, 77, 326, 338] not only because of the behavioural information they provide, but also because of the semantic information on the input types. This semantic information (in the form of an ontology) allows ontology-based test data generation. However, this semantic information does not necessarily code *realism*, nor can it always guarantee to do this.

4.7.2 Test Data Generation Approaches Using Existing Resources

Test cases can be generated using existing data, such as existing test cases and recorded user session data. The use of session data for web application testing has been suggested by several researchers [3, 163, 199].

Thummalapenta et al. [315] propose an approach that aids test generation approaches in reaching desired states. The proposed approach mines code bases and extracts sequences that are related to the object types of the method under test.

However, the only approach that aims to generate test data for ScS was proposed by Conroy et al. [74]. In this approach test cases are generated using the data from applications with Graphical User Interfaces (GUIs). The approach harnesses data from GUI elements and uses the harnessed data to generate test cases for ScS. Compared to approaches presented in Section 4.7.1, this approach might be expected to be more effective in generating realistic test data. Unfortunately, however, the proposed approach remains unautomated.

McMinn et al. [211] also proposed automatic generation of realistic test data, aiming to reduce the human oracle cost. The authors propose the use of genetic algorithms combined with data and input domain knowledge to increase the fault-finding capability, and the branch coverage probability of the generated test data. The approach also includes re-using of test data among related units of the system under test. Even though this approach is capable of generating 'realistic' test data, as with most of the automated approaches it can only provide structural realism.

Alshahwan and Harman [4] use search based testing to generate branch adequate test data for server side code in PHP. Their approach was shown to be effective at branch coverage. The proposed approach could be used to seed our approach for generatable input data as a base case. This remains a topic for future work.

4.8 Conclusion

In this chapter, we introduced a novel approach that can automate realistic test data generation. The proposed approach not only effectively generates realistic inputs, it also minimises the dependence on existing data. Another major benefit of this approach is its ability to produce customised and realistic test data. The test results obtained in the experiments provide evidence for the effectiveness of our approach compared to the state of the art automated test data generation approach. In both case studies, our approach comfortably outperformed random test data generation.

Chapter 5

Cost Reduction Through Multi-objective Data Source Selection

5.1 Motivation

Testing is one of the most widely used and important ways in which software engineers gain confidence in systems' behaviour and with which they find faults. Testing is important for all kinds of software systems, but this chapter is concerned with web service based systems. Trust is considered as one of the technical barriers' to enterprises transition to such ScS [64]. One of the potential solutions for establishing trust among different stakeholders is testing, which potentially provides the necessary assurance in correct functioning of ScS. Not surprisingly, this pressing need has led to a dramatic recent increase in the number of publications on ScS testing, as discussed in Chapter 2.

Service oriented systems present many challenges to the tester. The services available for use can change repeatedly, frequently and unpredictably, while many may offer substantially similar services, but with different quality and performance attributes and at different costs. For the tester, this presents the challenge of a rapidly changing, non-deterministic system with important choices to be made about the costs involved in testing; not merely execution costs, but monetary costs accrued from charges for third party service use.

In additional to the inherent complexities of web service testing, there is also a need to construct realistic test cases [3, 74, 163, 200, 210, 211]; test data that achieves fault revelation and which does so with test cases that are achievable in practice and understandable to the human tester and user alike. For instance, it has been argued that user session data is valuable precisely because it is real data [3, 163, 200]. Other authors have also developed testing approaches to harness realism in testing, drawing realistic data from the Graphical User Interfaces of the systems under test [74] and from 'web scavenging' for suitable test data [210, 211].

Our approach to the challenge of testing web services is to seek to exploit the flexibility and, more specifically, the composability of services as a key mechanism for finding suitable solutions. Essentially, we seek, not only to ameliorate problems in web service testing, but to turn these problems around using these very same demanding characteristics as the basis for potential solutions. Our approach recommends composition topologies that deploy existing services to generate realistic tests.

In the previous chapter, we introduced the tool ATAM, which implements this approach. ATAM finds services that combine to form a composition topology (an acyclic graph in which nodes are services and edges denote input-output connectivity between each service). We demonstrated that this approach has the potential to help automate the process of finding realistic test data.

ATAM recommends a set of candidate topologies from which a user can select a suitable choice, resulting in an approach to generate realistic test data for the service under test. However, this does not completely solve the tester's problem. There may be many services available from which to choose for each node of the topology. Since services may come at a cost and may offer differing degrees of reliability, there is an inherent multi-objective optimisation problem underpinning the final choice of services to select in order to concretise the topology. This choice is to balance the cost-benefit trade off.

In this chapter, we study this problem as a bi-objective typed selection problem, in which services of the right types for each node in the composition topology must be selected to balance testing cost against the reliability of the overall composition selected. We used NSGA-II to investigate the effect of three factors (composition size, composition topology and the number of services discovered) on performance (computation time) and quality (approximation to the pareto front). We used real world data on price to inform the cost choices and the results from previous chapter to determine the topologies to consider. However, we have no real-world reliability data available. Therefore, we study the effects of various models of the relationship between cost and reliability (logarithmic, linear and exponential) and various degrees of correlation strength (stronger, medium, weaker) between cost and reliability.

The primary contribution of this chapter as follows:

1. We introduce a multi-objective solution to service-centric test data generation. This approach is the first to apply multi-objective optimisation to service-based testing.

2. We present an investigation of the behaviour of our approach using different price-reliability models. We confirm that NSGA-II performs almost optimally on those problems for which problem size is sufficiently small to support exhaustive search for the globally optimal pareto front. On larger problems, we find that size and topology have more effect on quality and performance than the number of service choices discovered. This is encouraging because it provides evidence that the approach may cope with (widely anticipated) significant increases in the number of services that will become available in the coming years.

The rest of this chapter is organised as follows. Section 5.2 discusses existing test data generation approaches that use optimisation techniques and the concept of QoS in SOA. Section 5.3 explains the details of the proposed multi-objective optimisation for our approach. Sections 5.4 presents our case studies, research questions and our method of investigation. Section 5.5 presents the results from our experiments and answers the research questions. Section 5.6 concludes the chapter.

5.2 Background

In this section, we discuss the approaches to test data generation that are most closely related to our approach and also provide a brief information on the concept of QoS in SOA.

5.2.1 Test Data Generation and Optimisation

Our approach formulates test data generation as application for multi-objective SBSE [132]. We seek service compositions that generate realistic tests of a given type using services not are necessarily designed for testing.

Multi-objective optimisation is not new to the test data generation domain. Though our work is the first to use a multi-objective approach to service-based testing. Lakhotia et al. [166], Oster and Saglietti [240] and Pinto and Vergilio [259] already proposed multi-objective test data generation. These approaches focus on structural coverage as the main objective and use additional objectives such as execution time, memory consumption and size of test set. There are other approaches, such as Sagarna and Yao [277] where branch coverage is formulated as constrained optimisation problem. Similarly, Alshahwan and Harman [4] used a single objective search-based testing approach to generate test data to achieve branch coverage of server side code in PHP. The use of multi-objective optimisations in test case selection has also been proposed. For example, Yoo and Harman [373] used objectives such as code coverage and execution cost in test case selection.

5.2.2 Quality of Service Models

Our approach leverages the ability to compose and orchestrate web services to achieve goals for which individual services may not have been defined. One of the advantages of SOA is the ability to discover and compare services with similar business logic. The ability to compare services is enabled by a well-known concept called QoS in SOA.

QoS requirements generally refer to several functional and non-functional qualities of services in SOA [44, 334]. According to Wan Ab. Rahman [334], the most common QoS characteristics are service response time, performance, reliability, execution price (price from here on), availability and security. The explanation given to four of these characteristics that are covered in this chapter are:

1. **Price**: The monetary cost of invoking a service. There might exist multiple price plans for services but ATAM only considers the pay per use option.

2. **Reliability**: The capability of maintaining the quality of service. We accept reliability as the percentage of invocations that maintained the expected QoS levels.

3. **Availability**: The presence of a service for invocation. We consider availability as the percentage of time that the service available for invocation.

4. **Response time**: The amount of time that a service takes to respond various requests. Response time might be provided in different ways, such as maximum or average time. In this research, we assume maximum response times are provided by QoS.

The need for a QoS model that covers necessary quality aspects of SOA are also addressed by industry. OASIS introduced a QoS specification called Business QoS (bQoS) [234] in 2010.

5.2.3 Pareto-Optimality and NSGA-II

Pareto efficiency is a concept in economics which is also applied to many other subjects such as engineering [239]. It is used in the optimization of a vector of multiple criteria by enabling the decision maker to investigate the trade-offs among optimal combinations of these criteria. Pareto frontier (or pareto front) is a set of solutions that are pareto efficient. A solution is accepted as pareto efficient when it is not dominated by any other solution in the solution space [120]. A solution s is said to dominate another solution s' if s are equal or better than s' in all attributes/objectives.

Non-dominated Sorting Genetic Algorithm-II (NSGA-II), is introduced by Deb et al. [81]. NSGA-II does not produce a single result, but provides a pareto front of best solutions by using elitism and pareto optimality. Elitism is maintained by ranking each of the discovered solutions after each generation according to nondomination level, as presented in Algorithm 1. NSGA-II also uses a mechanism called crowding distance, which helps achieving a wider pareto front with distant solutions [54]. In order to achieve a wider front, the mechanism measures the distance of each solution from the rest of the population and assigns a higher fitness value to more distant solutions (maintaining a crowding distance among individuals). The execution of NSGA-II is described in Algorithm 1 in more detail.

Algorithm 1 NSGA-II Algorithm [54]

```
1: procedure NSGA-II ($\mathcal{N}', g, f_k(x_k)$)    ▷ $\mathcal{N}$ members evolved $g$ generations to solve $f_k(x)$
2: Initialize Population $\mathbb{P}'$
3: Generate random population - size $\mathcal{N}'$
4: Evaluate Objective Values
5: Assign Rank (level) Based on Pareto dominance - sort
6: Generate Child Population
7:      Binary Tournament Selection
8:      Recombination and Mutation
9: for $i = 1$ to $g$ do
10:    for each Parent and Child in Population do
11:        Assign Rank (level) based on Pareto - sort
12:        Generate sets of nondominated vectors along known pareto front
13:        Loop (inside) by adding solutions to next generation starting from the first front until $\mathcal{N}'$ individuals found determine crowding distance between points on each front
14:    end for
15:    Select points (elitist) on the lower front (with lower rank) and are outside a crowding distance
16:    Create next generation
17:         Binary Tournament Selection
18:         Recombination and Mutation
19: end for
20: end procedure
```

Deb et al. [81] also introduced a distance metric (Υ) which is used for measuring how close the discovered set of solutions are to the optimal front. In order to measure this metric, the minimum Euclidean distance between each of the solutions from the generated set S' and the optimal front (or the optimal set S) is calculated and the average of these distances is used as the distance metric.

The Euclidean formula for the distance between two points $p = \{p_1, p_2, ..., p_n\}$

and $q = \{q_1, q_2, ..., q_n\}$ in Euclidean n-space is:

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{5.1}$$

The distance metric is formulated as:

$$\Upsilon = d(S', S) = \frac{1}{m} \sum_{i=1}^{m} (d_{min}(S'_i, S)) \tag{5.2}$$

where m is the number of solutions in S', $d_{min}(i)$ denotes the minimum Euclidean distance between the ith solution in S' and S.

5.3 Multi-objective Service-centric Test Input Generation

In this section, we explain the necessary elements required to support replication of results for the multi-objective optimisation approach we propose here: the objective function and genetic operators. We also discuss the choice of our algorithm and parameter values selected for the genetic algorithm.

5.3.1 Objective Functions

We introduce three different functions for the four QoS parameters we intend to include in the next version of ATAM: price, reliability, availability and response time. The reason for having three different objective functions is caused by our perception of combined reliability, availability and response time for compositions.

The objective function for the price parameter is straightforward. The function is the sum of the costs of all services required in a topology. In our approach, we considered price as an objective rather than a constraint, in order to allow the tester to explore all the possible solutions on the pareto-optimal front. The following is introduced as the objective function for minimising total cost of test data generation:

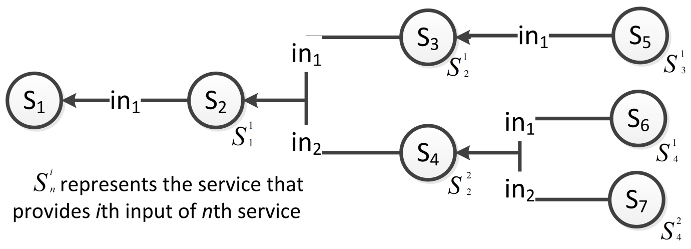


Figure 5.1: Example topology scenario for generating realistic test data using services. In the figure, each node in the topology represents a service description. In this scenario, service S_1 is the service that provides the required test input. The services on the other end, such as S_5, S_6 and S_7, are services that either require tester input or automatically generated input using a predefined algorithm. Each edge in the figure represents an input required by the service that is targeted by the edge. The source of the edge is the service that provides the required input. For example, in this scenario, the input for S_1 is provided by S_2.

$$Minimize \sum_{i=1}^{n} p_{s_i}$$

where n is the total number of services in the selected topology and p_{s_i} is price of the ith service.

The objective function for the other three QoS characteristics are not as straightforward. The total reliability, availability or response time of a topology is as high as the combined value of each characteristic for all the services used in that topology. We introduced the concept of 'combined characteristic value' (referred to as combined value from this point on) because the reliability, availability and response time of an individual service is not solely defined by the behaviour of that service in isolation. For example, in the case of reliability, the reliability of a service S also depends on the reliability of the services that generate the necessary input for S. Figure 5.1 illustrates an example solution and explains necessary concepts in combined value calculations.

The same formulation can be used in the calculation of combined values for reliability and availability. For response time, however, we provide a third formulation. Due to space constraints, we explain our formulation for reliability and availability focusing on the reliability only. The combined value for reliability (referred to as combined reliability (cr)) of a service is formulated as:

$$cr(S_n) = r_{S_n} \times ir(S_n)$$

where $cr(S_n)$ is the combined reliability and r_{S_n} is the reliability of the service S_n and $ir(S_n)$ is the reliability function that calculates the combined reliability of the services that provide inputs for S_n.

The reliability calculation for inputs ($ir(S_n)$) varies based on the number of services providing the input. This is because in our approach, a service in the composition can get the required inputs in two possible manner:

case1 *From the tester or predefined algorithm*: In this case, the input reliability of the service is accepted 100% reliable (i.e. reliability score = 1.0). Services S_5, S_6 and S_7 in Figure 5.1 are examples to this case.

case2 *From some arbitrary number of services*: In this case, the input reliability of the service is equal to the lowest of the combined reliability of the services that provide its inputs. For example, service S_3 takes input from service S_5 while S_4 takes input from two services S_6 and S_7 as depicted in Figure 5.1.

In the light of these definitions, we formulated our input reliability function to suit these two cases as follows:

$$ir(S_n) = \begin{cases} 1.0 & \text{if } S_n \text{ is case 1} \\ MIN(cr(S_n^1), cr(S_n^2), \dots, cr(S_n^{in(S_n)})) & \text{if } S_n \text{ is case 2} \end{cases}$$

where S_n^i is the service providing ith input for service S_n and $in(S_n)$ is the total number of inputs service S_n has.

The reliability score of a composition is equal to the combined reliability of the first service (service at the highest level). In light of the given combined reliability calculation, the objective function for maximising the total reliability is formulated as:

$$Maximise\ r_{S_1} \times ir(S_1)$$

Having services that require inputs from multiple services allows ATAM to invoke the services which provide the inputs in parallel. Due to this parallel invocation ability, response time needs to be calculated in a similar fashion to reliability and availability.

We formulated combined response time as:

$$cres(S_n) = res_{S_n} + ires(S_n)$$

where $cres(S_n)$ is the combined response time and res_{S_n} is the response time of the service S_n and $ires(S_n)$ is the response time function that calculates the combined response time of the services that provide inputs for S_n.

There is also a minor difference in combined response time calculation. Even though the cases (case1 and case2) are also valid for response time, the values for the cases are different. The function that suits these two cases for response time is as follows:

$$ires(S_n) = \begin{cases} 0 & \text{if } S_n \text{ is case 1} \\ MAX(cres(S_n^1), cres(S_n^2), \dots, cres(S_n^{in(S_n)})) & \text{if } S_n \text{ is case 2} \end{cases}$$

where S_n^i is the service providing ith input for service S_n and $in(S_n)$ is the total number of inputs service S_n has.

We believe the testers use expected response time as a means to minimise the time it takes to generate test cases. As a result, the objective function needs to minimise the response time.

$$Minimise\ res_{S_1} + ires(S_1)$$

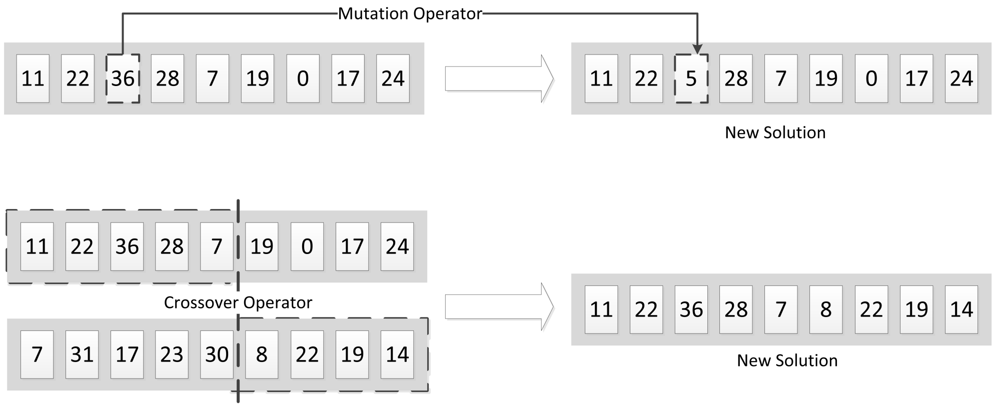


Figure 5.2: Illustration of the mutation and the crossover operators

5.3.2 Representation and Genetic Operators

After discovering all possible topologies, ATAM asks the tester to select one topology for test data generation. Then ATAM applies the optimisation process only to the selected topology using the service groups for this topology.

ATAM was implemented in a way to facilitate genetic operators (mutation and crossover) with service groups and topologies. As a result, in our multi-objective solution, genomes are represented as an array of integers as depicted in Figure 5.2. Each element in the array represents a 'service group' and values in each element represent a service that belongs to that group. The numbering of service groups and web services are based on the order of their discovery. For example, if the tester selects one of the topologies from Figure 5.2, the values of the elements in each genome represent the services which are in the places of $S_1, S_2, ..., S_9$ based on their order in the genome.

The initial population is generated using the number of services in each service group that form the selected topology. Each solution in the initial population is generated by randomly assigning a number to each array between 0 and the number of services in the service group which is represented by the element.

The mutation operator modifies the value in an element with a number between 0 and the number of services that the group contains. The crossover operator produces a new solution by combining two solutions into a new solution as depicted in Figure 5.2.

5.3.3 Mutli-Objective Algorithm and Parameters

We selected the NSGA-II as optimization technique for our problem due to its reported performance against other algorithms for similar problems [387]. We used a modified version of the popular ECJ framework [92] which provides a built-in NSGA-II algorithm, and integrated it to ATAM. After some tuning, we found out that the ideal parameters that provide the most diverse solutions for our problem are 35% mutation probability for each gene and one-point crossover with 50% crossover probability.

5.4 Empirical Studies

In this section, we introduce the case studies we used in our experiments, present the research questions we asked and explain our method of investigation.

5.4.1 Case Studies

Not having real-world case studies is one of the major set backs in SOA testing research [44]. As expected, we were unable to find existing services with measures of suitable QoS values. The only QoS characteristic that the authors have access to is service cost. As a result, we selected a group of existing commercial services with publicly available pricing details (presented in Table 5.1) which are collected from the Remote Methods website [268]. Due to access restrictions on non-commercial users, we were unable to verify the accuracy of the provided prices. We were also unable to verify continual service availability.

For 7 out of 9 groups in Table 5.1, we were able to identify at least two existing services with price details as a basis for determining the maximum and minimum prices. For groups 3 and 4, we were unable to find multiple services. However, for these two groups, we used the prices from different subscription models of the same service. In some of the groups, outputs of free to use services might not be the same as the paid ones (due to the level of detail provided), however, we accepted these services as alternatives to the paid ones due to their similar service descriptions.

Service Group		**Price (per query)**			
No	**Description**	**Max**	**Company**	**Min**	**Company**
1	Phone verification	$0.300	StrikeIron	Free	WebServiceMart
2	Traffic information	$0.300	MapPoint	Free	MapQuest
3	Geographic data	$0.165	Urban Mapping	$0.010	Urban Mapping
4	Bank info verification	$0.160	Unified	$0.090	Unified
5	IP to location	$0.020	StrikeIron	Free	IP2Location
6	Stock Quote	$0.020	XIgnite	$0.008	CDYNE
7	Financial data	$0.017	Eoddata	$0.007	XIgnite
8	Nutrition data	$0.010	CalorieKing	Free	MyNetDiary
9	Web search	$0.005	Google	Free	Bing

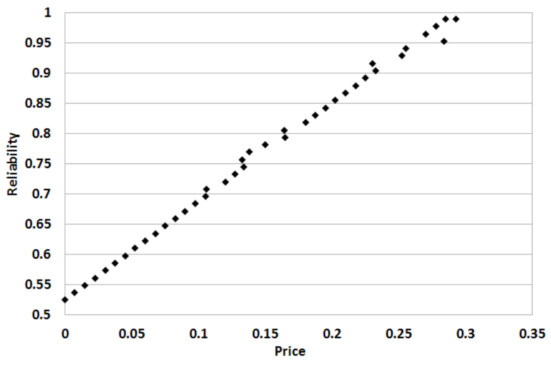
Table 5.1: Services used as a basis for the synthetically generated case study. The services and the given prices in this table are collected from Remote Methods website [268].

In our case study, we focused on two QoS characteristics: cost and reliability. The reason for choosing cost is due to existence of real-world references. We chose reliability due to our better understanding of the concept compared to availability and response time.

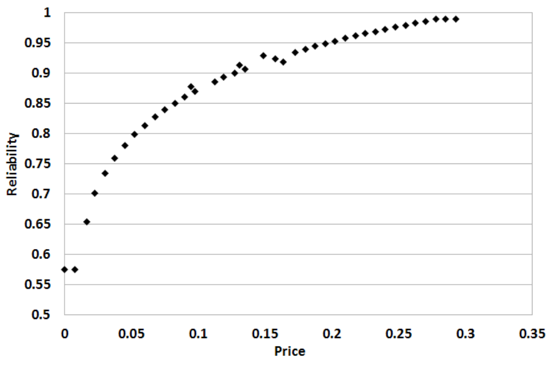
At present, making a realistic projection on the relation between reliability and cost is a challenging task due to lack of real-world reliability data. To compensate, we defined three price-reliability models (referred to as model) that construe the relation between price and reliability as illustrated in Figure 5.3.

We started generating our case study with the assumption that reliability and price are positively correlated. The reliability value for each service in each group cover is between 0.50 to 0.99. The cost value for each service in each group is assigned using the minimum and maximum prices given on the Table 5.1. We generated 9 service groups with 40 services in each group for all 3 models initially. Data for other group sizes (20 and 30) are generated by removing service entries from the initially generated groups.

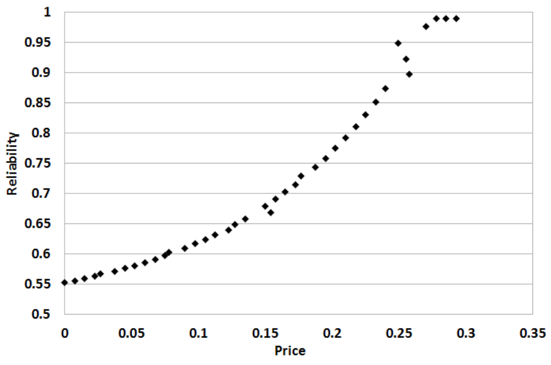
Composition size in the experiments represents the number of groups (starting from Group 1) that are included in a given topology. For example, composition size 4 means a composition which includes the first 4 groups.



(a) Linear model



(b) Logarithmic model



(c) Exponential model

Figure 5.3: Example reliability-price distribution of services in a group (Group size: 40 services) for all three models (with three very strong correlation).

5.4.2 Research Questions

We ask the following four questions:

RQ1 Is there a relation between the used price-reliability model and the discovered pareto front?

RQ2 How is discovered pareto front affected by different levels of price-reliability correlation in the models?

RQ3 What are the effects of parameters such as composition size, group size and topology complexity on the discovered pareto front?

RQ4 What are the effects of parameters on performance?

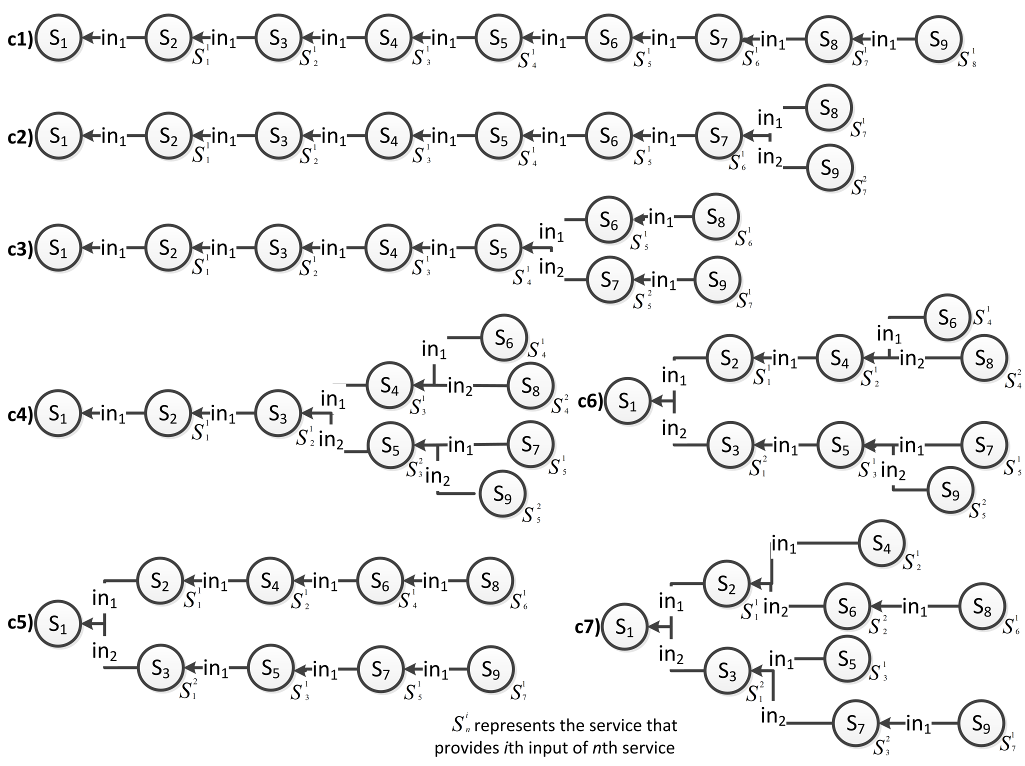

c1)
c2)
c3)
c4)
c5)
c6)
c7)
S_n^i represents the service that provides *i*th input of *n*th service

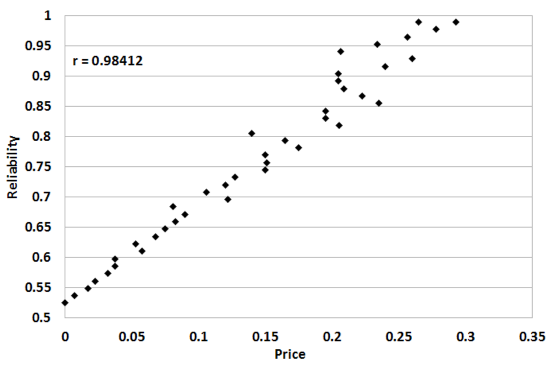
Figure 5.4: Illustration of the topologies that are used in order to answer RQ2 and RQ4.

5.4.3 Method of Investigation

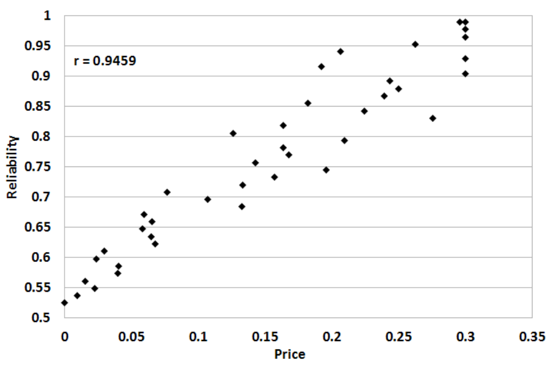
In order to answer RQ1, we generated pareto fronts for all three models with most of the possible combinations (allowed by our case study) using different group sizes (20, 30 and 40) and composition sizes (3, 4, 5, 7 and 9). Then we investigated the fronts generated from these groups.

In order to answer RQ2, we created new case studies by introducing 3 different level of correlation (low, medium and high) in each model and investigated the fronts generated from these case studies. Figure 5.5 illustrates these three levels of correlation on the linear model.

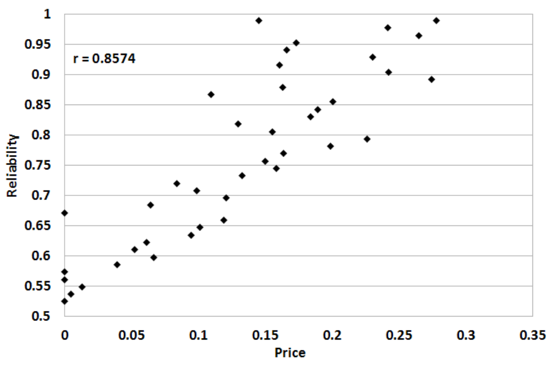
In order to answer RQ3 and RQ4, we developed 7 different topologies with different complexities as depicted in Figure 5.4. First, we investigated the fronts generated from these topologies to answer RQ3. We also compared the execution times of the generated topologies using different population size and number of generations in order to answer RQ4.



(a) High



(b) Medium



(c) Low

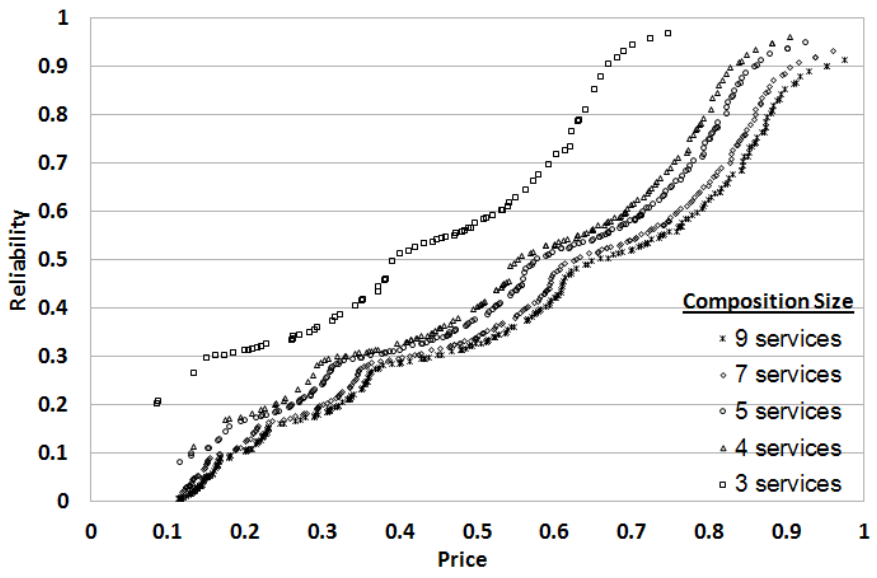
Figure 5.5: Different levels of correlation for the linear correlation model. The calculated Pearson's correlation coefficients for each correlation level from low to high are 0.8574, 0.9459 and 0.98412. The original distribution for this group is depicted in Figure 5.3a.

5.5 Results and Analysis

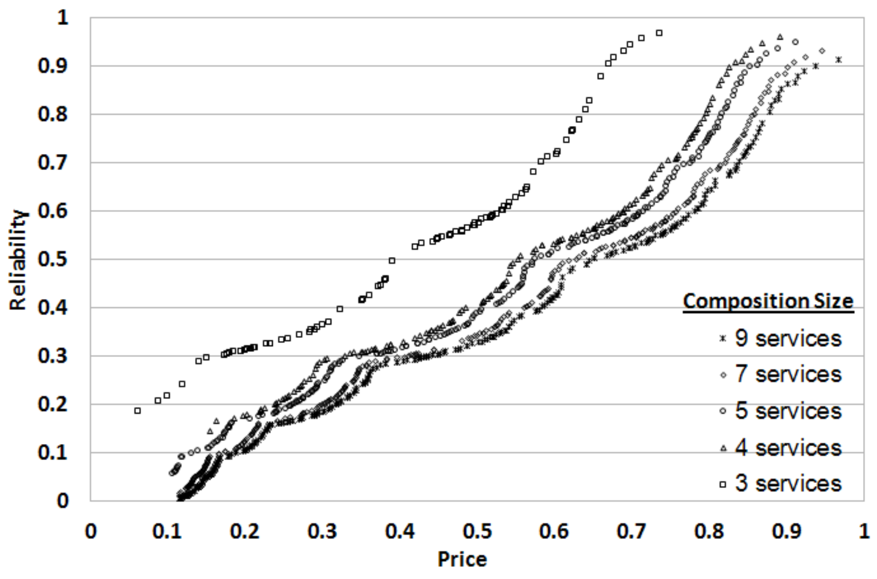
In this section, we present the results from our experiments, provide answers to the research questions we asked and discuss the threats to the validity of the experiments we conducted.

5.5.1 Results

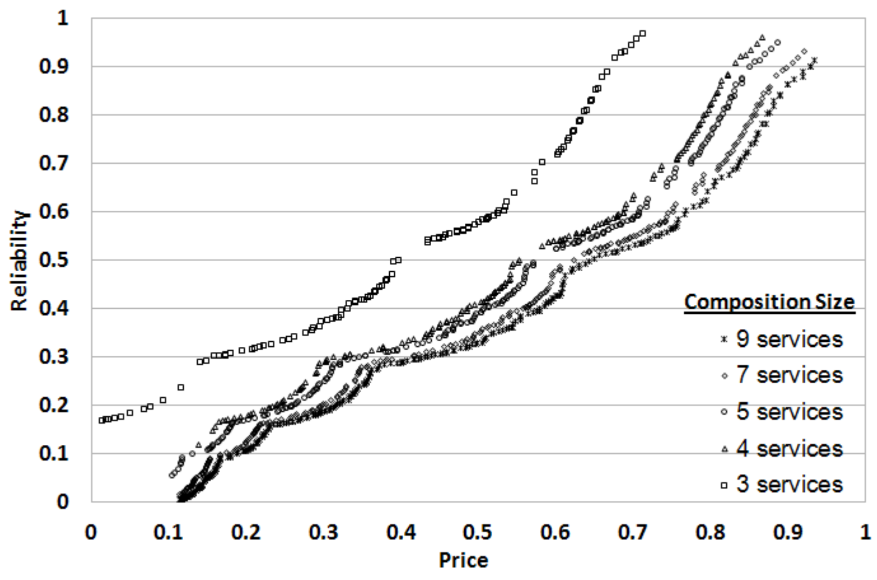
The discovered pareto fronts for all models are depicted in Figure 5.6 and 5.10 (using the topology complexity (c1) in Figure 5.4). As can be observed from these figures, the discovered fronts are very closely related to the model used. This similarity can be also observed in Figure 5.10.



(a) Group size: 20, Model: Exponential

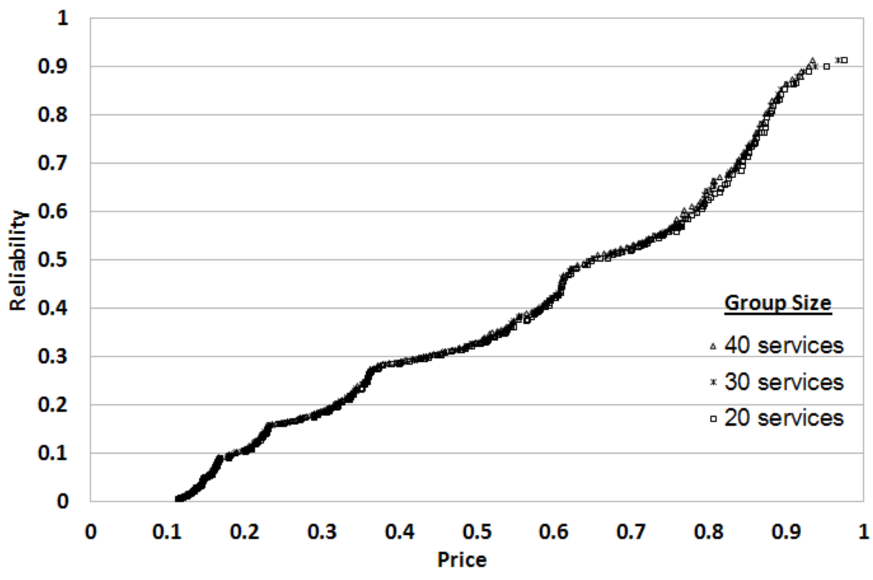


(b) Group size: 30, Model: Exponential

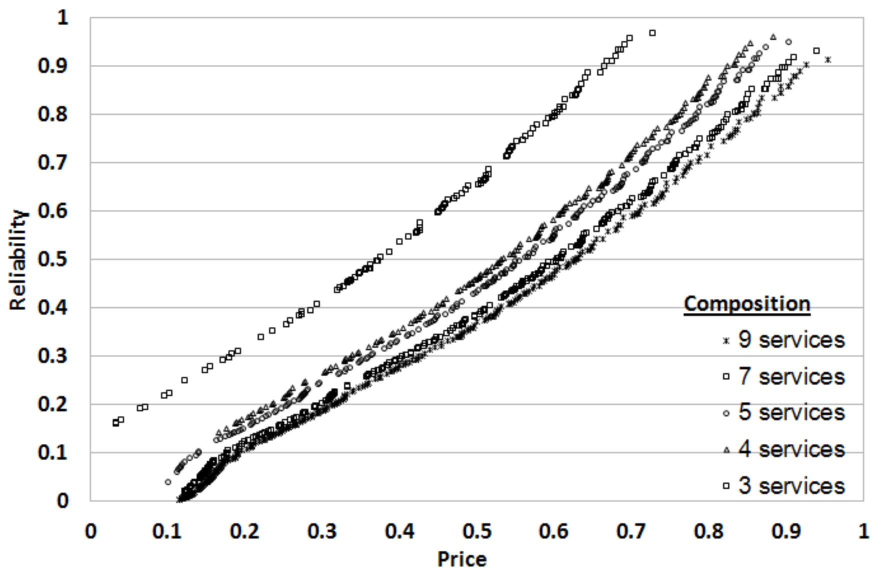


(c) Group size: 40, Model: Exponential

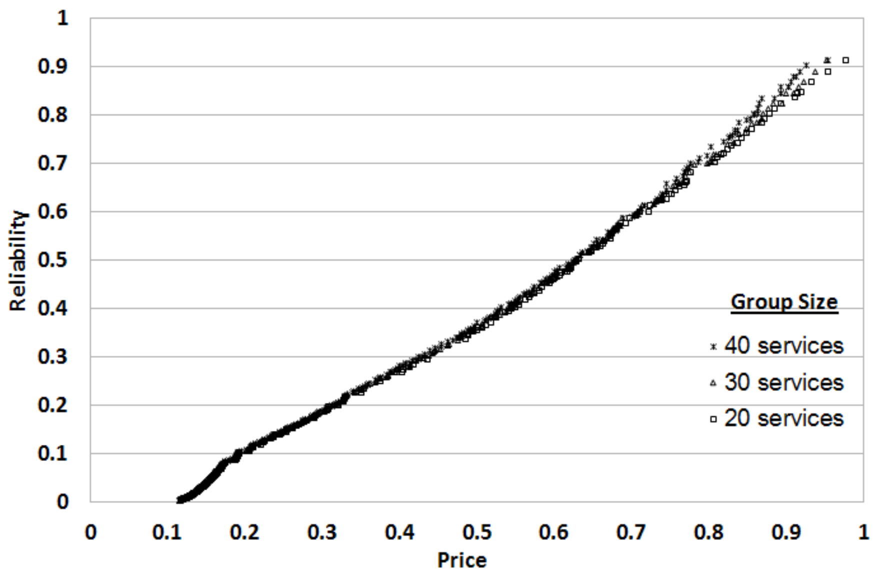
Figure 5.6: Continued on next page...



(d) Sequence size: 9, Model: Exponential

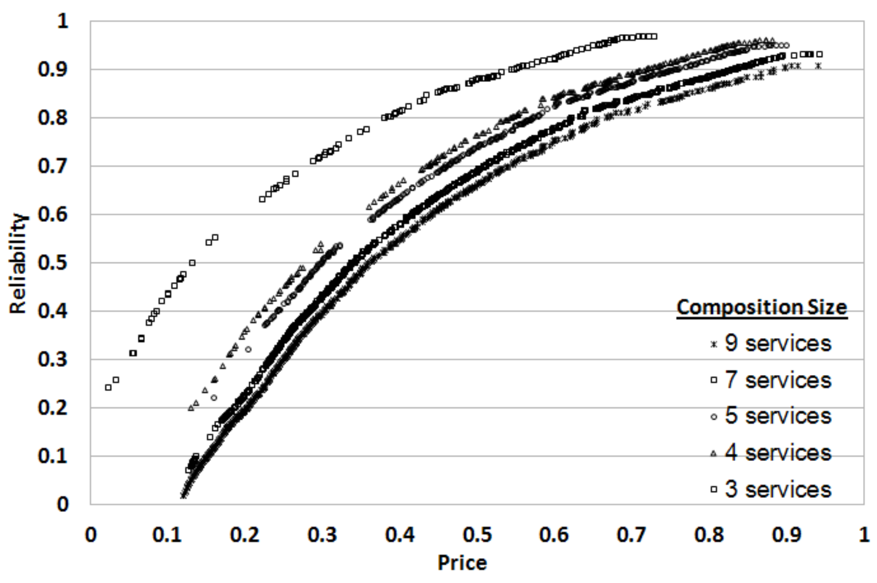


(e) Group size: 40, Model: Linear

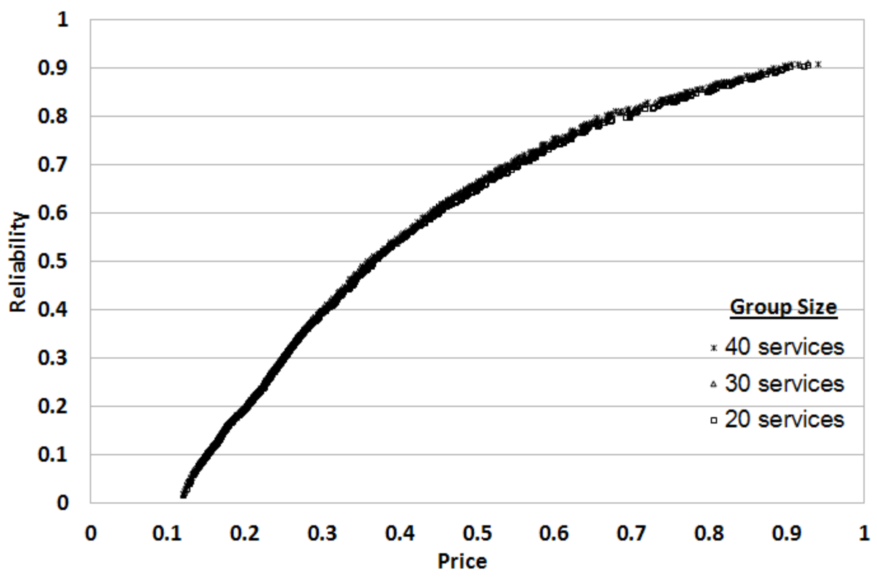


(f) Sequence size: 9, Model: Linear

Figure 5.6: Continued on next page...



(g) Group size: 40, Model: Logarithmic



(h) Sequence size: 9, Model: Logarithmic

Figure 5.6: The effects of the parameters used in our experiments on the generated pareto-front. Composition size represents the number of service groups that form a topology. Group size represents the number of web services in a service group. Model represents price-reliability relation.

The results suggest that the group size and composition size do have a minor effect on the overall form of the discovered pareto front. Composition size, as expected, causes a shift on the position of the discovered front. The results provide evidence for the fact that the observed shift is closely related with price of the added/eliminated service group in the composition. As depicted in Figure 5.6g, the biggest shift is observed between composition sizes 3 and 4, due to the relatively high price of the services in group 4. The effect of the reliability in this case is less observable due to the relatively low difference between reliability scores of services in each group.

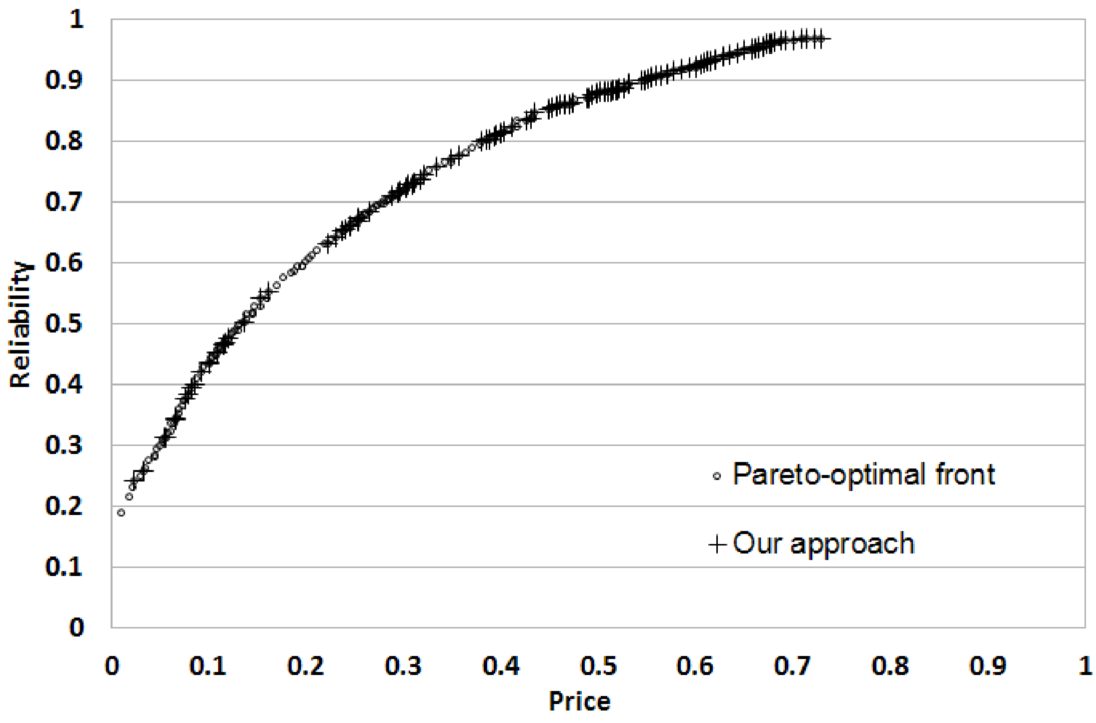
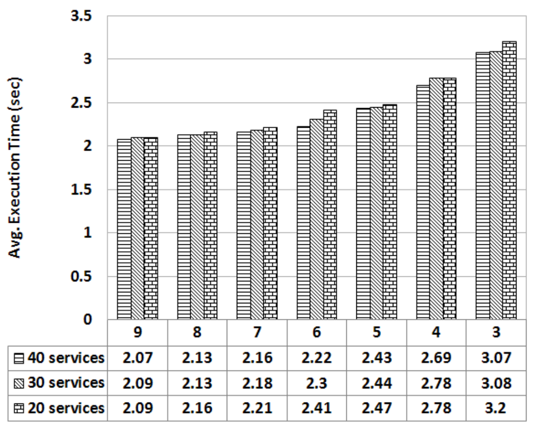


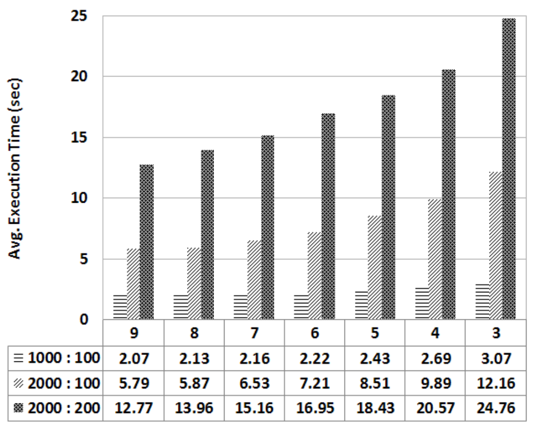
Figure 5.7: The difference between the globally optimal front and the front discovered by our approach. Measured distance (Υ) between fronts is ~ 0.00024 price-reliability points.

Search-based optimisation techniques are often applied to problems with large solution sets due to their effectiveness in such scenarios. In our problem, the search space is not fixed and it can vary greatly (based on the composition and group sizes). We investigated the performance of our approach in small search spaces in order to compare to known globally optimal fronts. For topologies small enough for exhaustive search, we found that NSGA-II finds a pareto front very near the true pareto front as depicted in Figure 5.7. The Euclidean distance between the optimal front and the discovered front is ~ 0.00024 price-reliability points. We also measured the average execution time of our approach and the time it takes to calculate the optimal pareto front in Figure 5.7. Exhaustive calculation of the optimal front took 4.5 seconds whereas our approach (using NSGA-II with population size 400 and 100 generations) took 0.8347 seconds.



	9	8	7	6	5	4	3
▤ 40 services	2.07	2.13	2.16	2.22	2.43	2.69	3.07
▧ 30 services	2.09	2.13	2.18	2.3	2.44	2.78	3.08
▦ 20 services	2.09	2.16	2.21	2.41	2.47	2.78	3.2

(a) Effect of group size on execution time



	9	8	7	6	5	4	3
▤ 1000 : 100	2.07	2.13	2.16	2.22	2.43	2.69	3.07
▧ 2000 : 100	5.79	5.87	6.53	7.21	8.51	9.89	12.16
▩ 2000 : 200	12.77	13.96	15.16	16.95	18.43	20.57	24.76

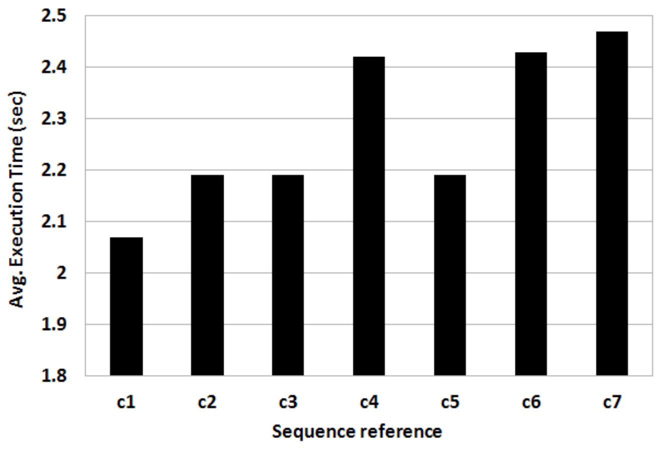
(b) Effects of population size and number of generations on execution time

Figure 5.8: The effects of the parameters used in our experiments on the execution time. The values in each graph represent the composition sizes. The average execution times for different group and composition sizes are provided in Figure 5.8a. The average execution times in Figure 5.8b are for different population sizes and number of generations (given in the format "population size : number of generations").

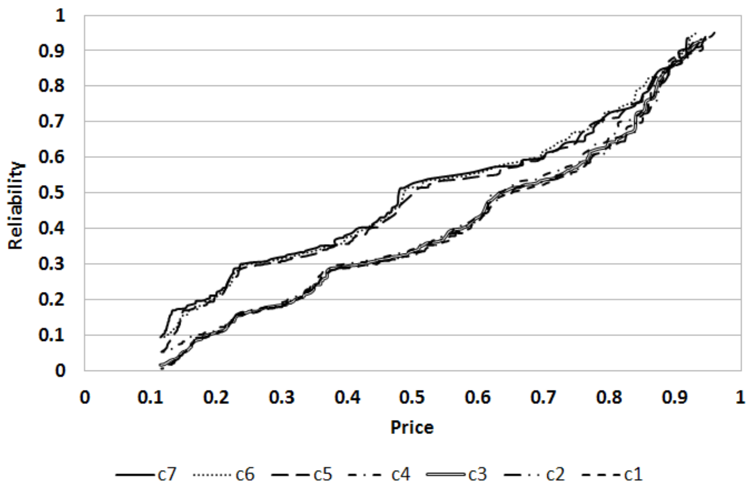
The effects of the parameters such as group size, composition size, population size and number of generations on the execution time are depicted in Figure 5.8. The results suggest that the effect of group size is negligible as depicted in Figure 5.8a. The performance difference between groups' sizes is only a fraction of a second for all composition sizes ranging from 0.02s up to 0.13s, whereas for almost all group sizes there is about 1s difference in execution times between composition sizes 9 and 3.

One unexpected effect which is very noticeable in both graphs is the negative correlation between the composition size and the execution time. We believe that this negative correlation is caused by ECJ's internal mechanism that maintains crowding distance. In order to prove our hypothesis, we measured the execution of the same case study with different population sizes and different number of generations. The results from these experiments are depicted in Figure 5.8b.

During our experiments, we also investigated the relation among composition size, number of generations and execution time. Our initial assumption was that the higher the composition size and the number of generations, the longer it should take to execute. The results provided evidence for the validity of our assumption. When we increased the number of generations from 100 to 200, execution time increased 2.21 times for composition size 9, whereas 2.03 times for composition size 3.



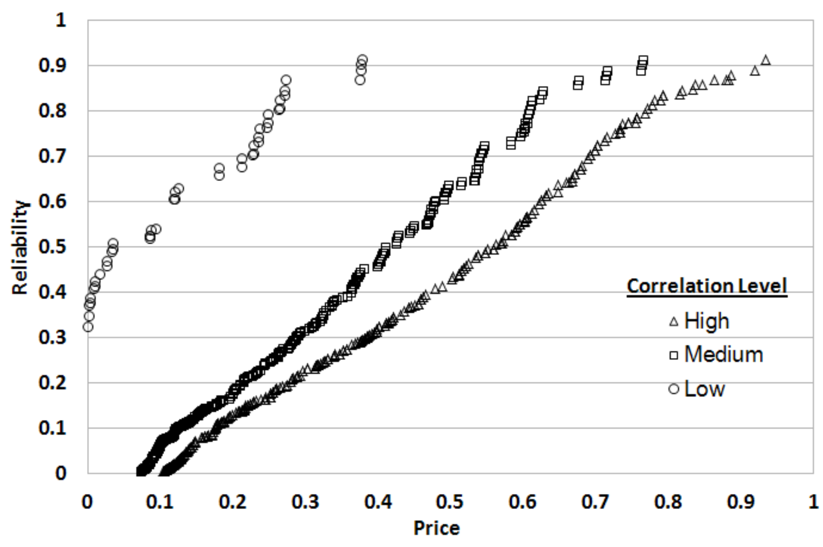
(a) Effect on execution time



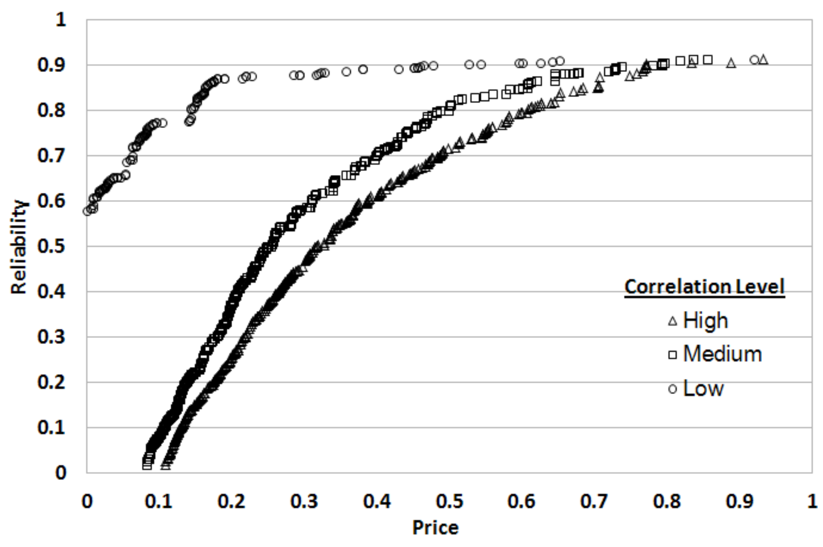
(b) Effect on pareto front form

Figure 5.9: The effects of topology complexity on pareto-front and execution time. The labels on the graphs ((c1) to (c7)) represent the results for the topologies in Figure 5.4. Pareto fronts in Figure 5.9b are presented in a different form (lines instead of individual solutions as points) than the previous figures in order to make these 7 pareto fronts more distinguishable.

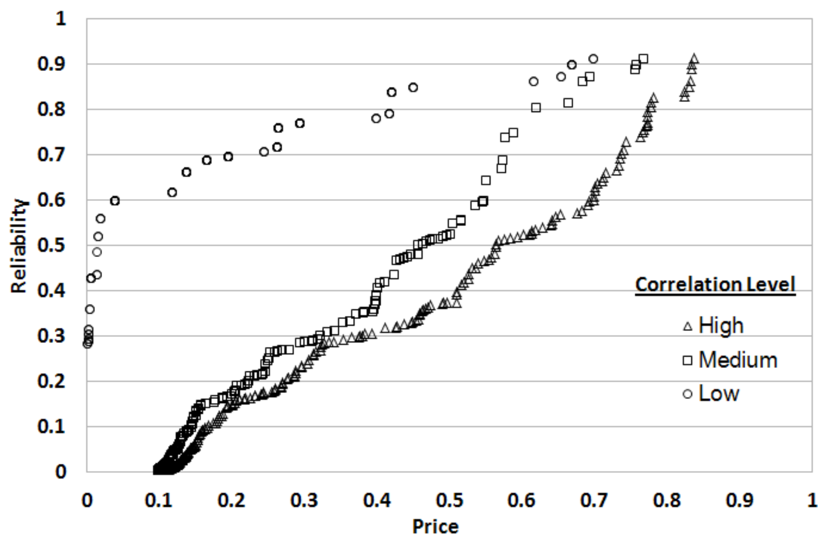
As mentioned, the results up to this point in the chapter are collected using the topology (c1) in Figure 5.4. The effects of complexity on execution time and pareto front are depicted in Figure 5.9. Investigation of the results in Figure 5.9a shows a positive correlation between the number of services that require multiple inputs and the execution time. In the case of topology (c1) the lowest execution time was achieved because this topology does not contain any services with multiple inputs, whereas the topologies (c4), (c6) and (c7) (topologies containing 2 multiple input services) have the highest execution times. Another important observation was that a topology's form has a negligible effect on execution times. For example, very similar topologies such as (c5) and (c7) have different execution times.



(a) Linear



(b) Logarithmic



(c) Exponential

Figure 5.10: Pareto fronts discovered from all three models (Composition Size: 9, Group size: 40 services) with different levels of correlation. The original distribution for this group is depicted in Figure 5.6.

Complexity is found to have the highest effect among all parameters on the from of the discovered pareto front. This effect can clearly be observed in Figure 5.9b, where the discovered fronts are separated into two distinct groups. The first group consists of topologies (c5), (c6) and (c7) and the second group consists of (c1), (c2), (c3) and (c4). The cause of this grouping is the similarities between the forms of the topologies in each group. The effect of complexity is almost negligible in higher reliability scores, whereas for the rest of the pareto front it is more visible. The fronts (c1) and (c7) in Figure 5.9b are a distance of ~ 0.073 price-reliability points apart.

The results suggest that correlation level in models have the highest effect on the form of the discovered pareto fronts as depicted in Figure 5.10. Comparison between the fronts of the original case study (Figure 5.6) and the modified ones show that there is a positive correlation between the correlation level and the number of solutions on the discovered front. Correlation level has the same effect on all three models as depicted in Figure 5.10. We believe that this effect is caused by NSGA-II tendency to generate solutions around the services with better QoS scores than expected in the model.

5.5.2 Answers to Research Questions

The results depicted in Figure 5.6 and 5.10 **answer RQ1** by providing evidence for the fact that the *cost-reliability relationship is replicated* in the discovered pareto fronts *when correlation is high*. However, as the price-reliability *correlation decreases, less smooth fronts with fewer solutions* are discovered.

The results also provide evidence for the fact that the level of correlation in a model can effect the form of the discovered pareto fronts. The results, which **answer RQ2**, suggest a *negative correlation between the model correlation level and the number of solutions on the discovered front*.

In order to **answer RQ3**, we have to provide a separate explanation for each parameter. The evidence from Figure 5.6h suggests that *group size has a negligible effect* on the discovered pareto front compared to composition size and topology complexity. As for the effects of the composition size, the results suggest that as one might hope, the *lower the composition size, the cheaper and more reliable the solution* will be (using the same set of services and topology). The results from Figure 5.9b suggest that among all the three parameters we investigated, *topology complexity has relatively highest impact*.

As for the **answer to RQ4** regarding the performance, we find that *composition size has the strongest effect*, with smaller topologies consuming more machine time; a curious effect we believe is due to the influence of crowding distance. The results suggest that *group size has a negligible effect.*

5.5.3 Threats to Validity

In the experiments, the most important external threat which limits the general applicability of the proposed approach is the test scenarios representing real-world ones. This concern is especially important for work in which experimental validation is performed using simulation and synthetic case studies. Although we performed our experiments using simulations in order to ensure maximum realism, we attempted to solve the realism problem on two fronts: realistic prices and realistic price-reliability models.

1. *Realistic prices*: Due to access restrictions and the high number of services needed to perform experiments, we could not use the actual prices for each service we simulated in the experiments. In order to make our simulation more realistic, we used the prices of existing services as a basis (maximum and minimum prices) for each service group. We believe the prices used in the experiments can serve a basis for a realistic scenario since they are real-world prices.

2. *Realistic price-reliability models*: Presenting a real-world model that represents price-reliability relation was one of the major challenges for the experiments presented in this chapter. To the best of the author's knowledge, there is no empirical study on this subject. Chapter 6 of this thesis includes an empirical reliability study (in Table 6.1) with existing services, however, the number of services analysed in this study were substantially less than the number of reliability scores needed for the experiments in this chapter. As a result, we chose the three com-

mon statistical models to represent the price-reliability relation. In addition to this, we also simulated three different levels of correlation for all three models. We believe that some of the combinations of these three models and correlations might represent real-world scenarios.

The only internal threat which might have affected the results obtained are the reliability and price scores assignment to each service. In order to maintain consistency in each group, we automated this process using three algorithms (for three models) to generate scores for a given number of services, with maximum and minimum prices. The algorithms are also set to generate reliability scores between 0.55 and 0.99, which we believe covers all possible real-world scenarios. The correctness of the algorithms was manually inspected.

5.6 Conclusion

In this chapter, we presented a multi-objective solution to service-oriented test data generation. We focused on the cost of test data generation and the reliability of the test data sources as our primary objectives. We chose a widely used multi-objective algorithm NSGA-II and investigated the behaviour of our approach in various situations, drawn from different models and parameters for our problem and solution domain. The results provided evidence for robustness of our approach.

Chapter 6

Cost Reduction Through Pareto-optimal Test Suite Minimisation

6.1 Motivation

As discussed in Chapter 2, one of the limitations of ScST is the cost associated with invoking services. Increased test frequency for ScS exacerbates the severity of the issues regarding testing cost. The cost of invoking services during testing is a major problem at composition level. Solutions aimed at reducing the cost of testing, such as simulated testing have previously been proposed [146, 186, 205, 208, 247, 270]. However, these approaches do not eliminate the need for runtime testing (testing with real services).

In the previous chapters, we addressed testing cost reduction at different levels of ScST. For example, ATAM (discussed in Chapter 4) aimed at reducing the cost of (manual) realistic test input generation through automation. In Chapter 5, we introduced another solution that aimed at reducing test input generation cost by using services with lower invocation cost. Although the work introduced in these chapters can help in reducing the testing cost in ScST, there remains the issue of controlling the test cost at runtime.

One widely studied solution aimed at reducing runtime testing cost is test suite minimisation [374]. The purpose of test suite minimisation and prioritisation is to reduce testing cost through removing redundant test cases. In test suite minimisation, there are concerns such as retaining coverage, test suite effectiveness or fault detection capability, execution time and resource usage, such as memory. According to the literature, problems where there are multiple competing and conflicting concerns can be investigated using pareto optimal optimisation approaches [133].

A tester is often faced with making decisions based on multiple test criteria during runtime testing. This is due to the fact that, in many situations, it is not expected that the tester aims to achieve a single goal during the testing process. According to the literature, one goal that is often used is code coverage. Achieving a high coverage is often regarded as a major goal in testing [361, 314]. However, achieving a high coverage is expensive and expecting to achieve 100% coverage might not be realistic when testing complex systems.

The cost of a test suite is one of the most important criteria, since a tester wants to get maximum value (e.g. number of branches covered) from the execution of a test suite [133]. In the literature, the cost of a test suite is often associated with the time it takes to execute it [374]. However, with the introduction of web services, service compositions and online APIs, the concept of monetary cost of testing is starting to gain acceptance in software testing literature.

The reliability (also referred to as quality) of a test suite is also an important concern. In Chapter 5, we introduced a formulation for the reliability score of a test input based on the reliability of its sources. We believe reliability is an important aspect especially in ScST, because it can help reduce the testing cost. A test suite with high reliability might reduce the cost of testing in two ways:

1. Human-oracle reduction due to the tester's trust in the test input source. If the test data used in testing has a high reliability, the tester will not need to inspect all test inputs for errors such as invalidity and conformance to input requirements (e.g. input format).

2. Avoiding extra testing cost due to erroneous test inputs. The extra cost in this case occurs due to service invocations with invalid test data during the execution and the tester's investigation of the unexpected test results that might occur as a result of invalid inputs.

For instance, consider the case of U.S. ZIP codes. Zip codes can be decommissioned or new ones might be assigned to new areas over time. As a result, during the testing of an ScS that requires ZIP codes as inputs, only valid codes must be used. In this scenario, if the ZIP codes (used in testing) are generated from unreliable services, then the test suite's reliability will be low, which means the test suite might include decommissioned or invalid ZIP codes. It is safe to assume that testing with decommissioned ZIP codes can be classified as unnecessary execution (except if the tester is performing robustness testing), which is likely to produce false positives. The results from our experimental studies for ZIP codes (discussed in Section 6.4) provided some evidence for the correctness of this assumption. The problem with false positives is that they might cause the tester to invest time in investigating the reasons for unexpected output(s).

In order to avoid the side effects of extensive runtime testing, a test suite minimisation approach could be used to remove 'redundant' test cases; those which merely cover previously covered features. On the other hand, to avoid side effects of a low reliability test suite, the selected approach must be able to manage multiple objectives. Thus, we propose to adapt the multi-objective test suite minimisation approach of Yoo and Harman [374] to ScST with the objectives, such as cost of service invocation, branch coverage and test suite reliability.

In this chapter, we introduce a pareto-optimal, multi-objective test suite minimisation approach to ScST aiming at reducing the runtime testing cost. The advantages of the proposed application of multi-objective test suite minimisation for ScS are:

1. Reduced cost in runtime testing through test suite minimisation.

2. Its ability to discover trade-offs between cost of test runs and system coverage.

3. Its ability to select a more reliable test suite without increasing the cost and affecting the coverage of the test suite.

The rest of this chapter is organised as follows. Section 6.2 briefly introduces the concept test suite minimisation and explains the proposed multi-objective test suite minimisation approach for ScST. Section 6.3 presents our case studies, research questions and our method of investigation. Section 6.4 presents the results from our experiments and answers the research questions. Section 6.5 concludes the chapter.

6.2 Multi-Objective Test Suite Minimisation for Service-centric Systems

In this section, we explain the concept of test suite minimisation, our approach and present our objective functions.

6.2.1 Test Suite Minimisation and HNSGA-II

Test suite minimisation (or test suite reduction) techniques aim to reduce the size of a test suite by eliminating redundant test cases [374]. Test suite minimisation is considered as a hitting set (or set cover) problem which is an NP-complete problem and defined as follows:

Input: A test suite T =$\{t_1, ..., t_n\}$ and a set of testing requirements R = $\{r_1, ..., r_n\}$ which need to be satisfied in order to provide the desired level of testing.

Goal: To find a representative set of test cases T_0, from T ($T_0 \subseteq T$) that satisfies all requirements. In order to satisfy all requirements, each r_i must at least be satisfied by one of the test cases that belongs to T_0. The effect of minimisation is maximised when T_0 is the minimal hitting set of the test cases in T.

Due to the test suite reduction problem (which is a minimal set cover problem) being NP-complete, the use of heuristics is proposed by many researchers [374]. According to the literature, another well known solution to the set cover problem is the greedy approximation. Yoo and Harman [372] proposed a solution (using an algorithm called 'HSNGA-II') which combines these two solutions into a pareto-optimal test suite minimisation approach. HNSGA-II uses the additional greedy algorithm, described in Algorithm 2 along with multi-objective algorithms.

Algorithm 2 Additional Greedy Algorithm ($\mathcal{U}, \mathcal{S}$) [373]

```
1: $C \leftarrow \emptyset$                                   ▷ covered elements in $\mathcal{U}$
2: repeat
3:     $j \leftarrow min_k(cost_k/|S_k - C|)$
4:     add $S_j$ to solution
5:     $C = C \bigcup S_j$
6: until $C = \mathcal{U}$
```

Where $\mathcal{U}$ is the universe, $\mathcal{S}$ is the set that contains $S_1, S_2, ..., Sn$ (with execution costs $cost_1, cost_2, ..., cost_n$) which cover subsets of $\mathcal{U}$, such that $\bigcup_i S_i = \mathcal{U}$. The assumption in this scenario is the existence of a subset of $\mathcal{S}$ which covers all elements of $\mathcal{U}$. The additional algorithm is cost cognisant, thus it does not just pick the subset that covers the most elements, but it aims at finding the subset that provides maximum coverage increase with the lowest cost at each iteration (at Line (4)) [373].

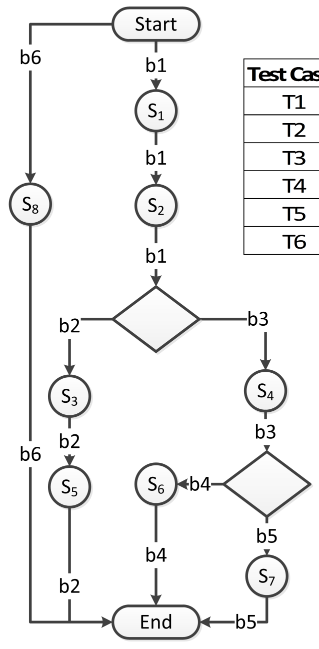
HNSGA-II is a variant of the standard NSGA-II and it may be more effective in multi-objective minimisation problems compared to NSGA-II. HNSGA-II combines the effectiveness of greedy algorithm for set cover problems with NSGA-II's global search. Results from the execution of additional greedy algorithm are added to the initial population of NSGA-II in order to create an initial population with better solutions compared to a 'random only' population. The goal in using a better initial population is to guide NSGA-II to a better approximation to the optimal pareto front. This process can be especially beneficial in problems with very large search spaces.

6.2.2 Proposed Approach

Our approach consists of two stages: test suite artefact calculation and multi-objective minimisation. After test suite generation, our approach requires the calculation of three measurements in order to apply multi-objective approaches. These are coverage, reliability and execution cost of each test case.

The reliability score of a test case is based on the reliability of its inputs. Reliability scores for generated inputs are provided by ATAM (details of reliability calculation for test cases were discussed in Chapter 5).

Unlike reliability, execution cost and branch coverage cannot be acquired from an external source. The easiest way of acquiring this information is by executing the whole test suite. Unfortunately, performing a runtime execution for the whole test suite in order to measure its artefacts will increase the overall cost of testing which is an unacceptable side effect for an approach that aims to reduce testing cost. In order to avoid this cost, we propose the use of simulated testing using mock/stub services as discussed in Chapter 2. Using stub/mock services will allow us to measure branch coverage and service invocation information for each test case without incurring additional costs.



Test Case	Reliability	Branches	Services	Cost
T1	0.90	b1, b2	S1,S2,S3,S5	$12.10
T2	0.87	b6	S8	$2.04
T3	0.91	b1,b3,b5	S1,S2,S4,S7	$17.15
T4	0.76	b1,b3,b4	S1,S2,S4,S6	$11.35
T5	0.99	b6	S8	$2.04
T6	0.93	b1, b2	S1,S2,S3,S5	$12.10

Figure 6.1: Example test suite reduction scenario for a ScS. The table depicts test cases in the suite with their reliability, branch coverage and execution cost calculated. For the given test suite (T1,...,T6) it is expected that test cases T1 and T2 will be eliminated to get the optimal test suite (T3,T4,T5,T6) which achieves 100% coverage with lowest cost and highest reliability.

Service invocation costs can occur in several ways (based on the type of contract agreed between the provider and the integrator). Two of the most prominent payment plans used at present are: pay per-use and invocation quota-based plans. As a result, two different cost calculation functions are introduced. Details of these two payment plans and the cost calculation associated with them are discussed in Section 6.2.3.

After completing stub/mock service generation, a simulated run for the test suite is performed in order to measure testing artefacts for each test case. An illustration of an example ScS and its test suite with the test case measurement is depicted in Figure 6.1.

In order to adapt Yoo and Harman's minimisation approach to ScST, we modified the original greedy algorithm by replacing its objective functions with the objective functions from our approach. We used the following algorithms for the 2-objective and the 3-objective optimisation scenarios. The additional algorithm for 2-objective optimisation (described in Algorithm 3) uses the cost and coverage calculation algorithms discussed in Section 6.2.3.

Algorithm 3 2-objective Additional Greedy Algorithm

```
Require: Test suite S
 1: DEFINE current fitness
 2: DEFINE test case subset $S' := \emptyset$
 3: while not stopping rule do
 4:     current fitness := 0
 5:     for all test cases in the test suite $S$ do
 6:         if test case $T$ is not in $S'$ then
 7:             current fitness := coverage score of $S'$ ($Cov_{S'}$)
 8:             ADD $T$ to $S'$
 9:             new fitness := $\frac{Cov_{S'} - \text{current fitness}}{\text{cost of } S'}$
10:         end if
11:         if new fitness is better than current fitness then
12:             current fitness := new fitness
13:             MARK $T$ as selected
14:             REMOVE $T$ from $S'$
15:         end if
16:         if No test case is selected then
17:             End
18:         else
19:             ADD $T$ to $S'$
20:         end if
21:     end for
22: end while
```

The 3-objective additional algorithm (described in Algorithm 4) considers an additional objective, reliability. In this algorithm, the objectives coverage, cost and reliability are combined into a single objective using the weighted-sum model (as depicted in Line(12)). In our experiments, both coverage and reliability objectives were given equal weights.

Algorithm 4 3-objective Additional Greedy Algorithm

```
Require: Test suite S
 1: DEFINE current fitness
 2: DEFINE test case subset $S' := \emptyset$
 3: while not stopping rule do
 4:     current fitness := 0
 5:     for all test cases in the test suite $S$ do
 6:         if test case $T$ is not in $S'$ then
 7:             coverage fitness := coverage score of $S'$ ($Cov_{S'}$)
 8:             reliability fitness := reliability score of $S'$ ($Rel_{S'}$)
 9:             ADD $T$ to $S'$
10:             coverage fitness := $\frac{Cov_{S'} - \text{coverage fitness}}{\text{cost of } S'}$
11:             reliability fitness := $\frac{Rel_{S'} - \text{reliability fitness}}{\text{cost of } S'}$
12:             new fitness := $\frac{\text{coverage fitness} + \text{reliability fitness}}{2}$   ▷ reliability and coverage are given equal weight
13:         end if
14:         if new fitness is better than current fitness then
15:             current fitness := new fitness
16:             MARK $T$ as selected
17:             REMOVE $T$ from $S'$
18:         end if
19:         if No test case is selected then
20:             End
21:         else
22:             ADD $T$ to $S'$
23:         end if
24:     end for
25: end while
```

As mentioned, HNSGA-II uses the results from the greedy algorithm runs as an initial population. In the second stage of our approach, we run the greedy algorithm and feed its results to NSGA-II algorithm (explained in detail in Chapter 5) which produces the pareto optimal front enabling the tester to investigate the trade-offs between the measured testing artefacts. In the cases where the size of resulting set from greedy algorithm is less than the required population size for NSGA-II, we compensate for this shortcoming by adding randomly generated solutions to the greedy results.

6.2.3 Objective Functions

Branch coverage is calculated as the percentage of branches of ScS under test covered by the given test suite. In our approach, we considered coverage as an objective rather than a constraint, in order to allow the tester to explore all the possible solutions on the pareto-optimal front. The following objective function is used for branch coverage since our aim is to maximise the coverage of the test suite.

$$Maximize \ \frac{\text{branches covered by test suite}}{\text{total number of branches}} \tag{6.1}$$

The objective function for the cost is not as straightforward as branch coverage because it has multiple options for service invocation cost. Several different payment plans might exist for service usage. However, we considered only the two prominent ones: pay per-use and quota based payment plans.

Pay per-use plan: In this case, the integrator is charged for each service invocation individually. The total cost of executing a test case is calculated as the total cost of services invoked by the test case and is formulated as:

$$cs(tc_m) = \sum_{i=1}^{n} X_{S_i} * C_{S_i}$$

where n is the number of services invoked by executing the test case tc_m, S_i is the ith executed service, C_{S_i} is the cost of invoking service S_i and X_{S_i} is the number of times service S_i invoked by this test case.

The cost for each service can be determined by discovering available services and their prices. At runtime, it is safe to assume that multiple alternatives for each service in the composition will be discovered. As a result, the tester might not know which services will be invoked at runtime. However, the tester needs to assign a static service cost to each service in the composition in order to calculate the cost artefact for each test case. In order to determine the price of each service, the tester might choose to use one of several criteria, such as using maximum, average or minimum price of the discovered services alternatives. This is flexibility essential because the tester might choose to change the runtime service selection criteria during the test runs (to force the invocation of low-cost services) in order to reduce the cost of testing. In this case, the lowest invocation costs for each service in composition is used to calculate cost of test cases. However, the tester might also choose to use the average or maximum invocation costs if a more realistic runtime testing is desired.

The objective function for pay-per use plan is formulated as:

$$Minimise \sum_{i=1}^{k} cs(tc_i) \tag{6.2}$$

where k is the total number of test cases and $cs(tc_i)$ is the total cost of executing the ith test case in the test suite.

Invocation quota based plan: In this case, the integrator pays a subscription fee for a number of service invocations within a period of time (such as monthly or annually). In our scenario, we presume that all the services used in the composition are selected from a single provider and a total invocation quota applies to all services rather than an individual quota for each service. The objective function for this plan is formulated as:

$$Minimise \text{ number of services invocations.} \tag{6.3}$$

Generating test data using ATAM also enables the use of another test case artefact: reliability. Reliability of a test case is based on the reliability of its inputs. Reliability of a test input is calculated by ATAM as the combined reliability of the data sources used in generation of this input. The reliability calculation and data source selection in ATAM are discussed elsewhere [43].

Each test case might include a combination of inputs generated using ATAM and user generated inputs. In the case of user generated inputs we consider the input to be 100% reliable and for ATAM generated inputs the reliability score is provided by ATAM. The reason behind considering the tester input as 100% reliable is our assumption of the tester's likely verification of the input data before test execution. Since we do not modify the tester inputs and use them as they are, we did not foresee any reason for having variations in the reliability score of the tester generated data.

In light of these possible cases, a reliability function covering these two cases is formulated as:

$$rf(in_x) = \begin{cases} 1.0 & \text{if } in_x \text{ is user generated} \\ \text{ATAM score} & \text{if } in_x \text{ is generated using ATAM} \end{cases}$$

where $rf(in_x)$ is the reliability score of the input in_x.

The reliability score of each test case is calculated as the average reliability of its inputs, and is formulated as:

$$rel(tc_m) = \frac{1}{y}\sum_{i=1}^{y} rf(in_i)$$

where y is the number of test inputs and $rf(in_i)$ is the reliability of the ith input (in_i) of the test case tc_m.

Reliability of a test suite is calculated as the average reliability of its test cases. Since our aim is to increase the reliability of the test suite, the objective function for test suite reliability is formulated as:

$$Maximise\ \frac{1}{z}\sum_{i=1}^{z} rel(tc_i) \tag{6.4}$$

where z is the number of test cases in the test suite and $rel(tc_i)$ is the reliability of the ith test case (tc_i) in the test suite.

6.2.4 Representation and Genetic Operators

Our aim is to find test case subsets of the test suite. We represented test suites and their subsets as bit vectors (as depicted in Figure 6.2), where each bit in the vector represents a test case. The bits that are set to 'true' represent the test cases that are to be included in the subset.

The mutation operator generates a new solution by modifying the value of a bit element in the vector. It replaces the current value of a bit element to the opposite value. The crossover operator produces a new solution by combining two solutions into a new solution as depicted in Figure 6.2. The illustrated operator is a single-point crossover operator, used in our approach.

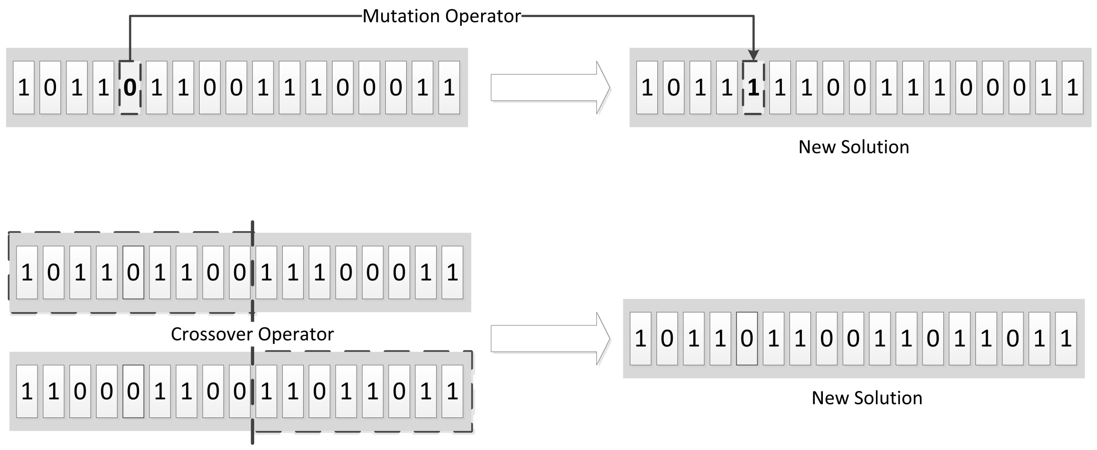


Figure 6.2: Illustration of the mutation operator and the crossover operators.

6.2.5 Mutli-Objective Algorithm and Parameters

To implement and evaluate our approach, we used the popular ECJ framework [92] which provides a built-in NSGA-II algorithm. We used a single population with a size of 2000 and set the number of generations to 100. After some tuning, we found that the ideal parameters that provide the most diverse solutions for our problem are: 5% mutation probability for each gene and single-point crossover with 90% crossover probability.

As mentioned, the only difference between HNSGA-II and NSGA-II is the initial population. The initial population of NSGA-II is generated by ECJ's internal mechanism. However, the initial population for HNSGA-II requires the use of additional greedy algorithms. The results from the additional greedy algorithm combined with the randomly generated solutions (in order to match the stated population size) are fed to EJC as the initial population of NSGA-II algorithm.

6.3 Empirical Studies

In this section, we introduce the case studies we used in our experiments, present the research questions we asked and explain our method of investigation.

6.3.1 Case Study

In order to evaluate our approach, we selected two case studies with different characteristics. The reason behind this selection is to observe the effectiveness of our approach in different scenarios by providing results that might challenge the findings from the other case study.

The first case study (CS1) is an example code used as a benchmark for applications/approaches that aim to achieve branch coverage called ‘Complex’. Complex is a artificial test object with complex branching conditions and loops [352]. However, in its original form, Complex is not an ScS. In order to evaluate our approach, we transformed Complex into an ScS by replacing all mathematical, relational and logical operators with service calls[1]. We choose an existing calculator web service [46] to replace the 4 mathematical operators: addition, subtraction, division and multiplication. For the other five operators, we implemented a web service providing the required services. The list of used services are presented in Table 6.4.

The second case study (CS2) is a synthetically created shipping workflow that combines the functionality of a number of available online services [105, 289, 330, 331]. In order to make CS2 as realistic as possible, the shipping service invokes a combination of existing web services and other synthetically created services. We were required to use synthetic services in order to simulate the functions of existing services that we have restricted access to.

The most important aspect of CS2 is that it works with real-world services, which requires realistic test inputs. The workflow requires a total of 14 inputs and 2 of these inputs are realistic (ZIP codes). The other inputs are user generatable shipping options such as mail type, delivery type, box weight and dimensions. The realism of CS2 is

[1]The source code available at http://www0.cs.ucl.ac.uk/staff/M.Bozkurt/files/public/complex_source.zip

also strengthened by the fact that the ZIP codes used in this study are generated from existing web services. The flow graph for the workflow is depicted in Figure 6.3 and the 14 web services invoked are presented in Table 6.5.

As discussed, one of the advantages of ATAM is its ability to generate test data based on the reliability of the test data source. In order to carry out our experiments, we needed to measure the reliability of the services we used for generating test inputs. However, this was not possible for some of the services we used due to access restrictions. In order to overcome this limitation and to increase the realism of our case studies, we need real-world reliability scores. Thus, we measured the reliability of 8 publicly available existing services, presented in Table 6.1.

Service	**Type**	**Number of errors**	**Reliability Score**
USPS.com	ZIP code validation	0	0.999
Websitemart.com	ZIP code validation	3102	0.744
Zip-codes.com	ZIP code validation	50	0.995
Webservicesx.com	ZIP code validation	727	0.939
NOOA.gov	Weather service	410	0.965
Myweather2.com	Weather service	146	0.987
CYDNE.com	Weather service	1218	0.899
Weatherbug.com	Weather service	550	0.954
Webservicesx.com	State ZIP code info	727	0.939

Table 6.1: The list of the services used in determining the real-world reliability scores used in our experiments. Services in the list are tested with 12171 ZIP codes belong to 23 U.S. States. The USPS service on the list is accepted as the ground truth for the validity of the ZIP codes. The rest of the ZIP validation services are evaluated with the generated ZIP codes to observe if they correctly identify given ZIP codes' validity. As for the weather services, we observed if they return weather information only for the valid ZIP codes.

There are three main reasons that led us to choose these services for this part of our experiments:

1. The public availability and having no access restrictions, evidently supporting replication.

2. Requiring the same input that can be validated using a service which can be accepted as the ground truth.

3. Similar functionality of services allowing determination of the expected output.

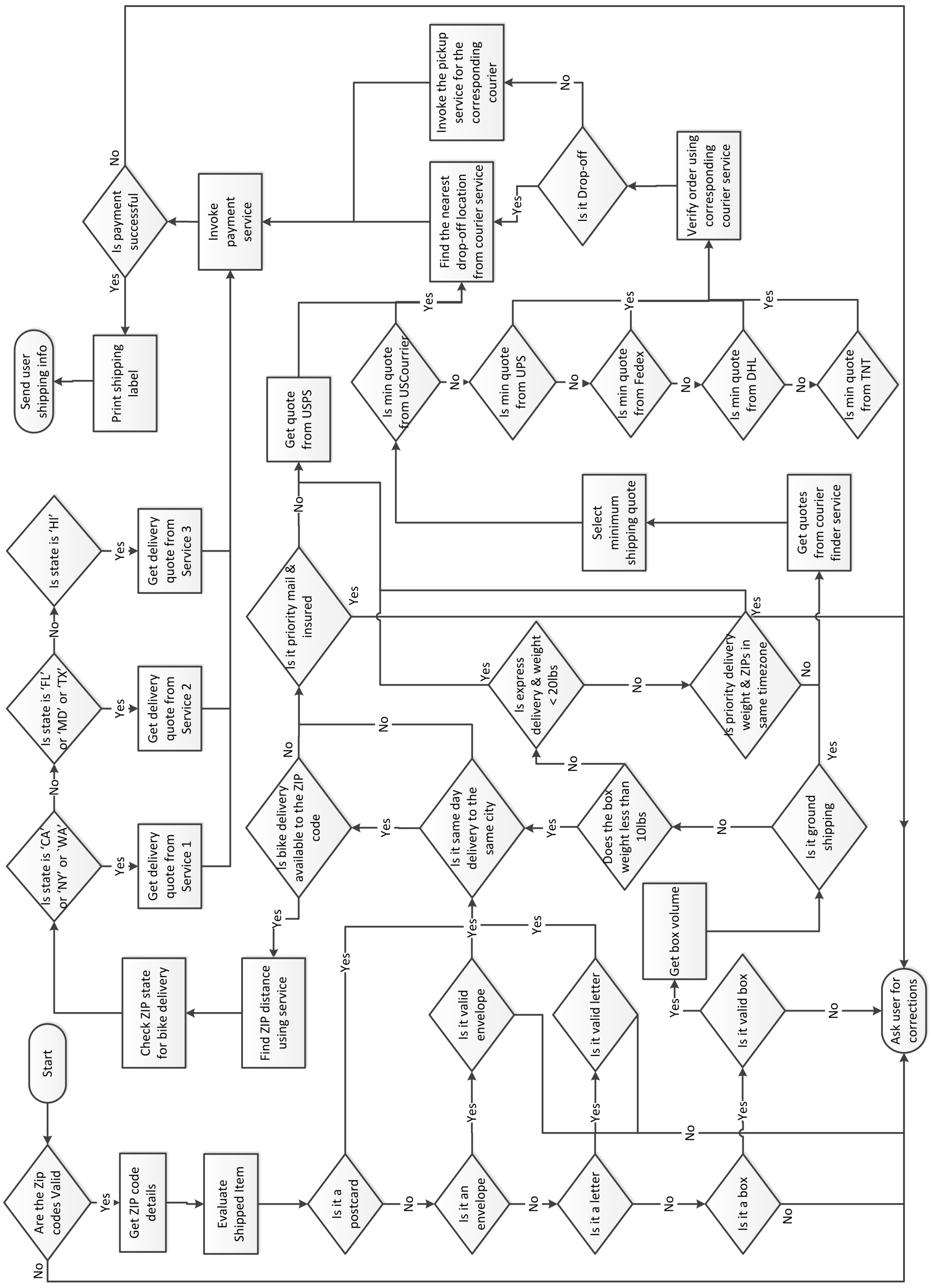

Start
Are the Zip codes Valid
Yes
No
Get ZIP code details
Evaluate Shipped Item
Is it a postcard
No
Is it an envelope
No
Is it a letter
No
Is it a box
No
Yes
Is it valid envelope
Yes
Is it valid letter
Yes
Is it valid box
Yes
No
Ask user for corrections
Get box volume
Is it ground shipping
Yes
No
Does the box weight less than 10lbs
Yes
No
Is it same day delivery to the same city
Yes
No
Is express delivery & weight < 20lbs
Yes
No
Is priority delivery weight & ZIPs in same timezone
Yes
No
Is bike delivery available to the ZIP code
Yes
No
Find ZIP distance using service
Check ZIP state for bike delivery
Is state is 'CA' or 'NY' or 'WA'
Yes
No
Is state is 'FL' or 'MD' or 'TX'
Yes
No
Is state is 'HI'
Yes
Get delivery quote from Service 1
Get delivery quote from Service 2
Get delivery quote from Service 3
Is it priority mail & insured
Yes
No
Get quote from USPS
Get quotes from courier finder service
Select minimum shipping quote
Is min quote from USCourrier
Yes
No
Is min quote from UPS
No
Is min quote from Fedex
No
Is min quote from DHL
No
Is min quote from TNT
Yes
Yes
Verify order using corresponding courier service
Is it Drop-off
Yes
No
Find the nearest drop-off location from courier service
Invoke the pickup service for the corresponding courier
Invoke payment service
Is payment successful
Yes
No
Print shipping label
Send user shipping info

Figure 6.3: Flowchart of the second case study

The services in the list are categorised into two main groups based on their functionality; ZIP code verification services and weather services. Verification services check if a given 5-digit US ZIP code is valid. Weather services provide current weather information for a given ZIP code.

Services in the list were tested using 12171 ZIP codes of 23 different U.S. states. The ZIP codes used in reliability analysis are generated using the the 9th service from Table 6.1. The USPS service in the table is accepted as the ground truth for the validity of the ZIP codes. As a result, the number of errors observed for this service is set to 0 and its reliability score is set to the highest reliability score: 0.999. The rest of the ZIP validation services are evaluated with the generated ZIP codes to observe whether they correctly identify the given ZIP code's validity. As for the weather services, we only considered the valid ZIP codes in order to maintain consistency and checked if they return weather information for all valid ZIP codes. For the reliability of the last service, we simply counted the number of invalid ZIP codes from the generated outputs.

Input	**Reliability**	
	Positive	**Negative**
A	0.999	0.744
B	0.995	0.939
C	0.965	0.987
D	0.899	0.954
E	1.0	1.0
F	1.0	1.0

Table 6.2: Reliability values used in the 3-objective evaluation of CS1. Positive and negative values for each input are assumed to be generated by a single web service with the given reliability on the list. The last two inputs are assumed to be human generated thus their reliability scores are 100%.

After acquiring the reliability scores, we generated test inputs for both case studies. As for the test inputs, Complex does not require realistic inputs, but requires 6 integer numbers which can be automatically generated. As a result, we did not use ATAM in generating test inputs for CS1. Instead, we generated the required test inputs using a random number generation method. However, in order to maintain consistency in our experiments, we also evaluated the 3-objective formulation of our approach on CS1. Thus, we needed to generate a fictional scenario where inputs are generated using different services such as the Random.org integer generator [267]. In this scenario, we assumed that the positive and negative values for each of the first four inputs (A to D) are generated using 8 different services. We assigned 8 of the measured reliability scores to these services as presented in Table 6.2.

For CS2 we determined 3 web service compositions, as presented in Table 6.3, to generate ZIP codes. The services and test inputs used in evaluating this case study are real-world entities. However, we were unable to measure the actual reliabilities of the three services in the first composition due to access restrictions. As a result, we assigned 3 of the reliability scores from Table 6.1 to the 3 services in the first composition (in Table 6.3) and used these scores in the combined reliability score calculation of this composition.

Service				Reliability	
	Publisher	**Input(s)**	**Output**	**Individual**	**Combined**
1	Google Search	Keyword	Search result	0.999	0.881
	Strikeiron	Http address	IP address	0.744	
	FraudLabs	IP address	US location	0.899	
2	Codebump.com	–	US state names	0.954	0.947
	Webservicesx	US state name	ZIP code	0.939	
3	Webservicesx	US area code	ZIP code	0.899	0.899

Table 6.3: Web service compositions used in generating ZIP codes. Individual reliability scores represent the scores for each service and combined scores represent the reliability of the composition. The combined score of a composition also determines the reliability scores of inputs generated using this composition.

The other needed artefact is the cost of invoking services. This raises the issue of how to choose realistic values for this characteristic. In Chapter 5, we discussed this issue of generating realistic invocation cost values for the services used in the experiments and presented a solution where realistic costs values are obtained using costs of existing services as a basis. We adopted the same approach again and used the cost values from Table 5.1.

The real-world web service we used in CS1 is a free-to-use web service and provides 4 of the services used in this case study. Unfortunately, the free services presented challenges for maintaining consistency especially on CS1 due to these services being the most invoked services. This problem had an impact on our experiments using a per-use payment plan, causing most of the test cases having the same execution cost even though there were large differences in the number of the services invoked. As a result, we assigned the invocation cost values presented in Table 6.4 to the services in CS1.

Service		**Price**
No	**Description**	
1	Multiplication	$0.300
2	Division	$0.300
3	Logical AND	$0.165
4	Logical OR	$0.160
5	Greater Than	$0.020
6	Less Than	$0.020
7	Equal To	$0.017
8	Subtract	$0.010
9	Add	$0.005

Table 6.4: The invocation costs for the services used in CS1.

For CS2, we used a combination of real service costs (for free to use services) and synthetically generated values (using the same method adopted in CS1) as presented in Table 6.5.

Service			**Price**
No	**Name**	**Description**	
1	WebserviceMart	ZIP code verification	Free
2	WebServicesx	ZIP code info	$0.090
3	Bike messenger 1	Find bike messenger for CA, NY, WA	$0.008
4	Bike messenger 2	Find bike messenger for FL, MD, TX	$0.007
5	Bike messenger 3	Find bike messenger for all other states	$0.007
6	WebServicesx	ZIP code distance	$0.020
7	USPS	Get delivery price	Free
8	Fedex	Get delivery price	Free
9	TNT	Get delivery price	Free
10	DHL	Get delivery price	Free
11	UPS	Get delivery price	Free
12	Payment system	get card payment	$0.30
13	Label system	print mail label	$0.017
14	Courier finder	find courier	$0.010

Table 6.5: Invocation costs for the services used in CS2.

6.3.2 Research Questions

We ask the following three questions:

RQ1 Can multi-objective test suite minimisation reduce the testing cost by finding optimal test suite subset(s) for ScST?

RQ2 Is using a test suite with low reliability (containing invalid inputs) likely to generate false positives which might increase the testing cost?

RQ3 Can HNSGA-II algorithm discover dominating solutions compared to NSGA-II and the additional greedy algorithm in our problem?

6.3.3 Method of Investigation

In order to answer RQ1, we applied our minimisation technique to both of the case studies. Initially, we tried to randomly generate a test suite that achieves 100% branch coverage for each case study. However, we experienced two issues while using random test generation. It was found to be ineffective in achieving full coverage in CS2 and many of the generated test cases cover exactly the same branches as another test case (equivalent test cases). For CS1 the test suite reached 100% coverage after 1000 test cases and for CS2, test cases covering 3 of the branches (where two ZIP codes of the same city are required) could not be randomly generated within several thousand tries. As a result, we manually generated 4 test cases that cover the uncovered branches in CS2 and applied a simple test case reduction technique to eliminate equivalent test cases.

The resulting test suite sizes were 50 test cases for CS1 and 100 test cases for CS2. The details of both test cases are presented in Table 6.6. We applied our approach to both of the test suites and measured the minimisation rates achieved by our approach for two different respects: reduction in the number of test cases and reduction in the test cost for both payment models.

Experiment	Test suite size	Coverage	Service invocations	Cost
CS1	50	100%	29760	$1302.90
CS2	100	100%	693	$212.45

Table 6.6: Details of the test suites generated for CS1 and CS2.

The main reason for seeking the answer to RQ2 is to investigate our assumption that there is a need for reliable test suite. In order to answer RQ2, we measured the false positives occurred during testing of the four weather information services. In this scenario, a false positive occurs when one of the four services returns a negative response to the given ZIP code which is identified as 'invalid' by USPS service.

In order to answer RQ3, we evaluated the generated pareto fronts from two different aspects. First, we ran greedy, NSGA-II and HNSGA-II[2] with all possible combinations of configurations (both payment plans and both number of objectives) in our approach for both case studies and performed a domination analysis to compare the performance of the algorithms. We also measured the distances between discovered pareto fronts to give us a better understanding of the differences between the fronts. The distance metric we used (Υ) is described in Chapter 5. In this part of the experiment, we used the pareto front generated by HNSGA-II as a reference front and measured the distance of the pareto front generated by NSGA-II to it.

6.4 Results and Analysis

In this section, we present the results from our experiments, provide answers to the research questions and discuss the threats to the validity of the results from experiments we conducted.

6.4.1 Results

We analysed the results of test suite reduction after the application of our approach to the initial test suites for both case studies in three different aspects. The first aspect is the reduction in number of test cases while retaining the branch coverage of the subject. As presented in Table 6.7, our approach achieved 84% reduction in the number of test cases for CS1 and 68% reduction for CS2.

During the experiments, we also found that only 12% of the test cases for CS1 and 34% of the test cases for CS2 in the initial test suites found out to be equivalent test cases. This is an important finding that justifies the diversity of the initial test suites, it

[2]We ran NSGA-II and HNSGA-II for 10 times in order to statistically evaluate their performance.

also provides evidence for the effectiveness of our approach, since the reduction rates are higher than the equivalent test case rates.

Experiment	Number of Test Cases		
	Test Suite	Our approach	Reduction
CS1	50	8	84%
CS2	100	32	68%

Table 6.7: Reduction in the number of test cases with the application of our approach. The results from our approach are the minimum number of test cases from the initial test suite that provide 100% branch coverage.

The second aspect is the reduction in the number of service invocations. This aspect relates to the testing cost when using a contract-based payment plan. Our approach in this case targeted for finding the minimum number of service invocations required to achieve 100% branch coverage. As presented in Table 6.8, our approach found a solution that reduces the number of invocations from 29760 to 415 for CS1 achieving 98.6% reduction while retaining the branch coverage of the test suite. As for CS2, our approach achieved a 72% reduction in the number of service invocations.

Experiment	Cost (contract based)		
	Test Suite	Our approach	Reduction
CS1	29760	415	98.6%
CS2	693	197	72%

Table 6.8: Reduction in the number of service invocations with the application of our approach. The number for test suite represents the number of service invocations performed by executing the initial test suite. The results from our approach are the minimum number of invocations necessary to achieve 100% branch coverage.

Experiment	Cost (per-use based)		
	Test Suite	Our approach	Reduction
CS1	\$1302.90	\$19.86	98.5%
CS2	\$212.45	\$65.99	69%

Table 6.9: Reduction in the number of service invocations with the application of our approach. The number for test suite represents the number of service invocations performed by executing the initial test suite. The results from our approach are the minimum number of invocations necessary to achieve 100% branch coverage.

The third aspect is the reduction in the monetary cost of testing. This aspect relates to the testing cost when using a per-use payment plan. Our approach in this scenario seeks to find the set of test cases with the least expensive execution cost and achieves 100% branch coverage. As presented in Table 6.9, our approach found a solution that reduces the total cost of testing from \$1302.90 to \$19.86 for CS1 achieving 98.5% reduction while retaining the branch coverage of the test suite. As for CS2, our approach achieved a 69% reduction in the total cost of testing.

A general assumption was that one might expect our approach to achieve the same or very similar reduction rates for contract-based plan and the per-use plan. However, the results from Table 6.8 and Table 6.9 suggest that this assumption might not hold in all cases. According to the results, in the case of CS1, the difference between the reduction rates is minor. However, for CS2 the difference is significantly higher.

In order to answer **RQ2**, we investigated the false positives generated by the 727 ZIP codes which are identified as invalid by the USPS web service. We tested all the weather services with invalid ZIP codes in order to observe whether they cause a false positive. In this scenario, a false positive occurs when a service does not return a weather information for an invalid ZIP code.

Service	**Type**	**False Positives**	**FP Rate**	**Error Rate**
NOOA.gov	Weather service	232	0.32	0.035
Myweather2.com	Weather service	1	0.0014	0.013
CYDNE.com	Weather service	493	0.68	0.101
Weatherbug.com	Weather service	196	0.27	0.046

Table 6.10: The list of false positives caused by erroneous test inputs. In this scenario, a false positive occurs when one of the four services return a negative response to the given ZIP code which is identified as 'invalid' by USPS service. Values in 'FP rate' column represent the false positive generation rate for invalid inputs. The values in 'Error rate' column represent the erroneous output generation rates for the test cases in the test suite.

As it is presented in Table 6.10, there is a big variance in the number of false positives generated by the weather services. We believe this variance might be caused by weather services having ZIP code databases (or using an external ZIP code service) from different sources. For example, 'MyWeather2' service might be using a database which is very similar to the service (Webservicesx) we used in generating the ZIP codes. However, for the other weather services, we observed a much higher false positive rate. For example, for the 'NOAA' service, the tester needs to check the test results for 232 ZIP codes and verify each of these ZIP codes if they used this test suite.

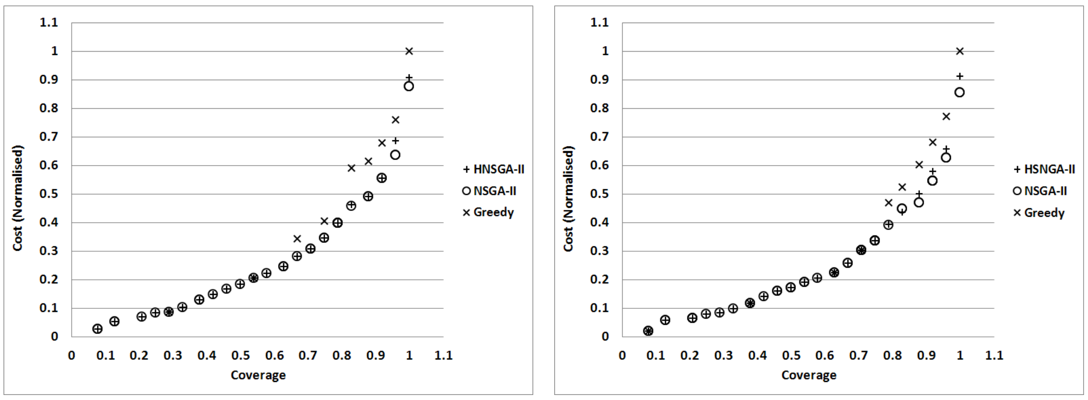
There is also another interesting result observed in Table 6.10: an invalid ZIP code is more likely to generate a false positive than a valid one to cause an erroneous output. For all services except one, the false positive rate is much higher than error rate.

One other important finding we need to discuss in this analysis is that none of the services generated the all 727 false positives we expected to observe. Only 561 of the invalid ZIP codes caused a false positive and for the remainder of the codes the weather services returned a valid response. We believe this is a further indication that supports our claim regarding weather services not using up-to-date ZIP code databases/services.

Experiment	Contract					Per Use				
	n	HNSGA-II		NSGA-II		n	HNSGA-II		NSGA-II	
		Avg.	σ	Avg.	σ		Avg.	σ	Avg.	σ
CS1 (2 obj.)	23.0	0.5	0.67	1.5	1.43	23.0	0.1	0.3	1.0	1.48
CS2 (2 obj.)	32.7	22.2	1.89	0.3	0.46	32.3	25.1	2.98	0	0
CS1 (3 obj.)	201.8	14.8	3.68	25.1	7.88	204.6	15.8	4.29	23.6	4.03
CS2 (3 obj.)	238.9	212	6.03	1.0	1.18	241	209	10.14	0.7	0.71

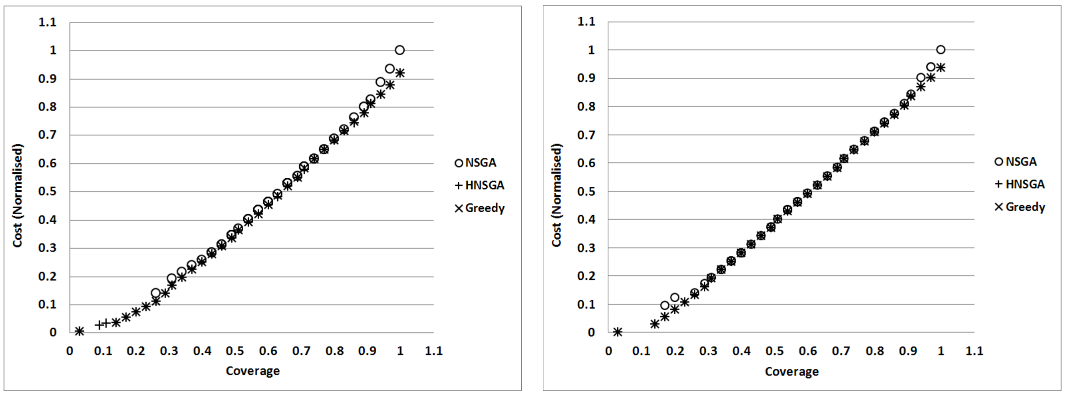
Table 6.11: Results from our domination analysis. The column 'n' represents the average size of the discovered pareto fronts. 'Avg.' column for each algorithm represents the average number of dominating solutions discovered by the algorithm. The results lead to two important findings that NSGA-II and HNSGA-II's performances are similar for problems in which greedy algorithm does not outperform NSGA-II (such as CS1). However, for problems (such as CS2) where greedy outperforms NSGA-II, HNSGA-II outperforms NSGA-II.

The results from the domination analysis (presented in Table 6.11) revealed results that conform with the analysis of Yoo and Harman [372]. Our first finding came from the results of CS1 that NSGA-II might outperform HNSGA-II (by a small margin) where additional greedy algorithm cannot outperform NSGA-II (as depicted in Figure 6.4, 6.7 and 6.9). We believe this is due to HSNGA-II starting with a good initial population, leading it to discover solutions around this set whereas NSGA-II explored a larger space using random population. However, as can be observed in Figure 6.4, 6.5 6.6 and 6.8, the majority of the solutions discovered by both algorithms are the same.



(a) Contract-based payment plan (b) Per-use payment plan

Figure 6.4: Pareto fronts discovered from the 2 objective optimisation of CS1



(a) Contract-based payment plan (b) Per-use payment plan

Figure 6.5: Pareto fronts discovered from the 2 objective optimisation of CS2

On the other hand, the results of CS2 provide evidence for the improvements HNSGA-II can provide for problems in which greedy outperforms NSGA-II. The results from both 2- and 3-objective runs indicate that HNSGA-II outperforms NSGA-II by a high margin. For example, on average, 68% of the discovered solutions with 2 objectives for CS2 (contract-based plan) dominate NSGA-II and the domination rate goes up to 89% for the same scenario with 3 objectives. A similar trend was also observed for other configurations of CS2.

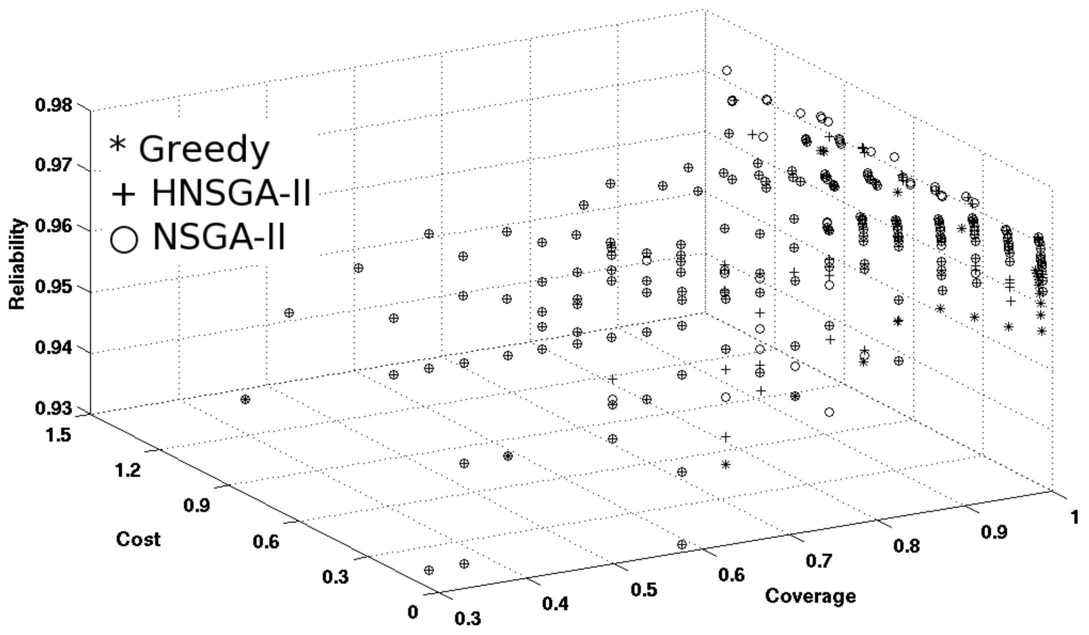


Figure 6.6: 3 objective optimisation for CS1 with per-use payment plan

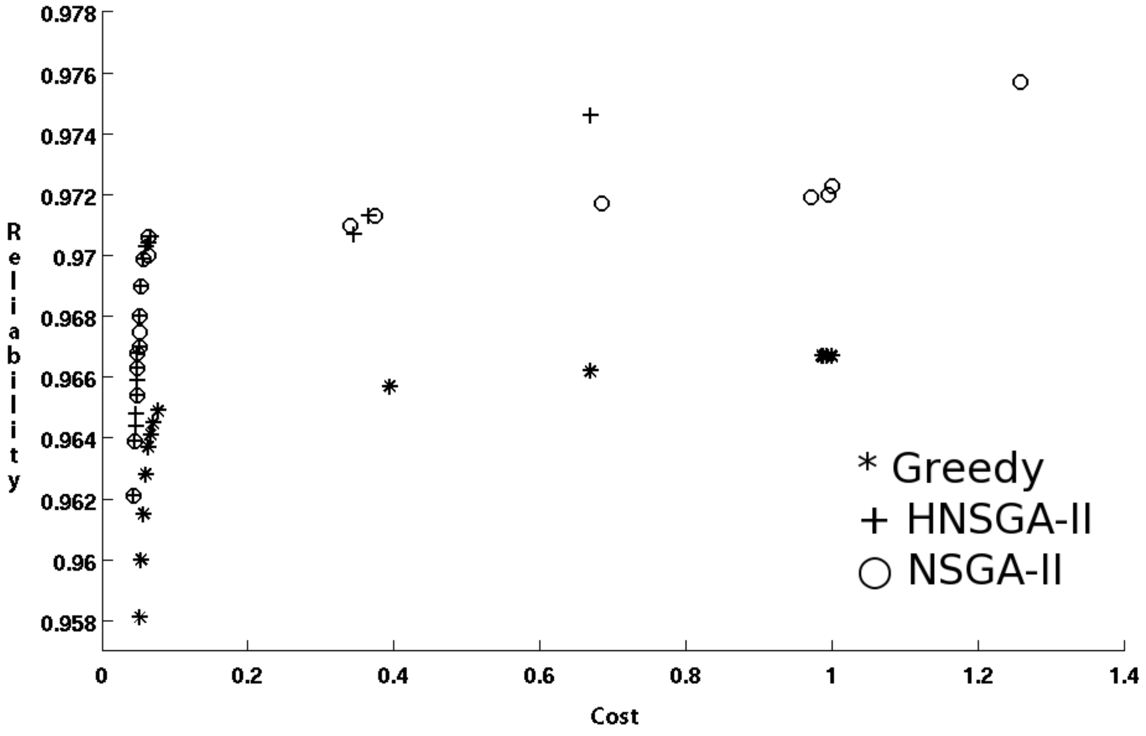


Figure 6.7: Projected view of Figure 6.6 focusing on the solutions with 100% coverage score.

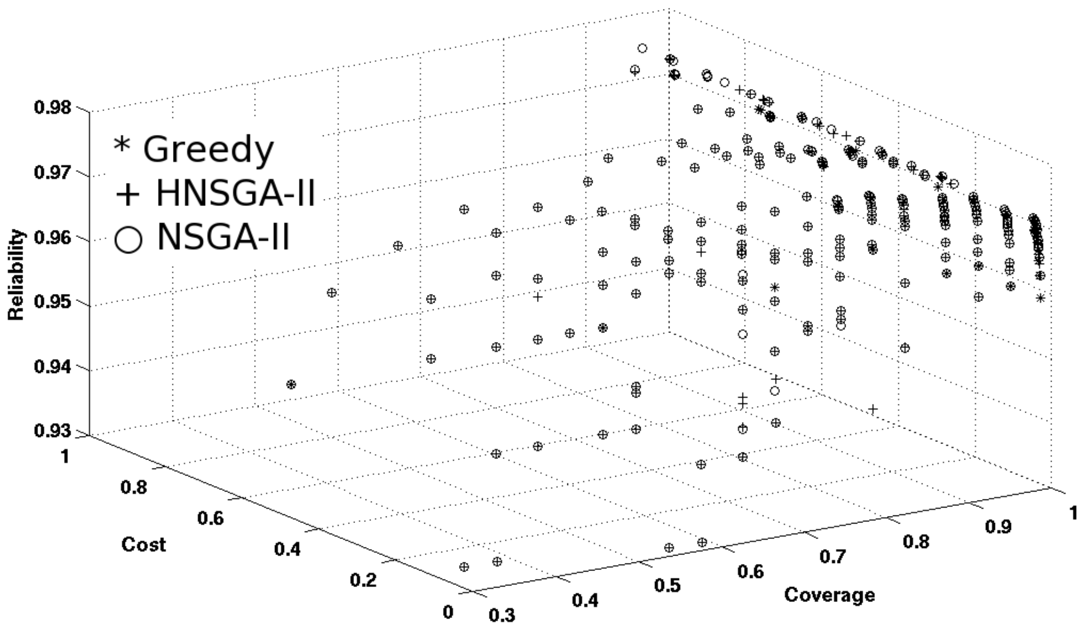


Figure 6.8: 3 objective optimisation for CS1 with contract-based payment plan

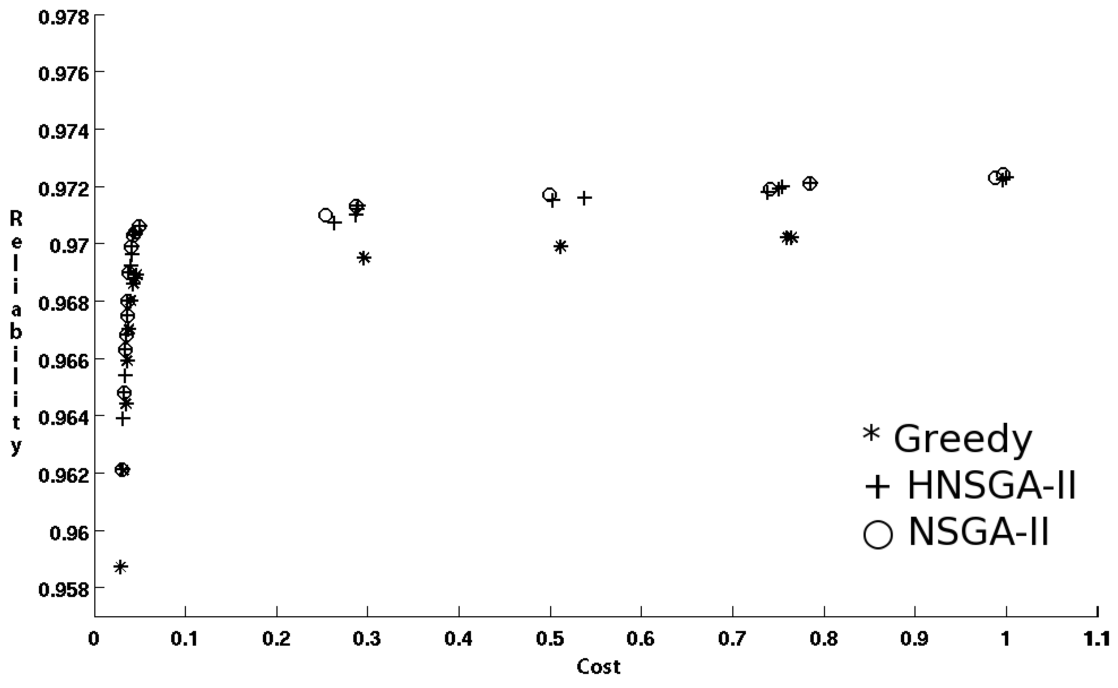


Figure 6.9: Projected view of Figure 6.8 focusing on the solutions with 100% coverage score.

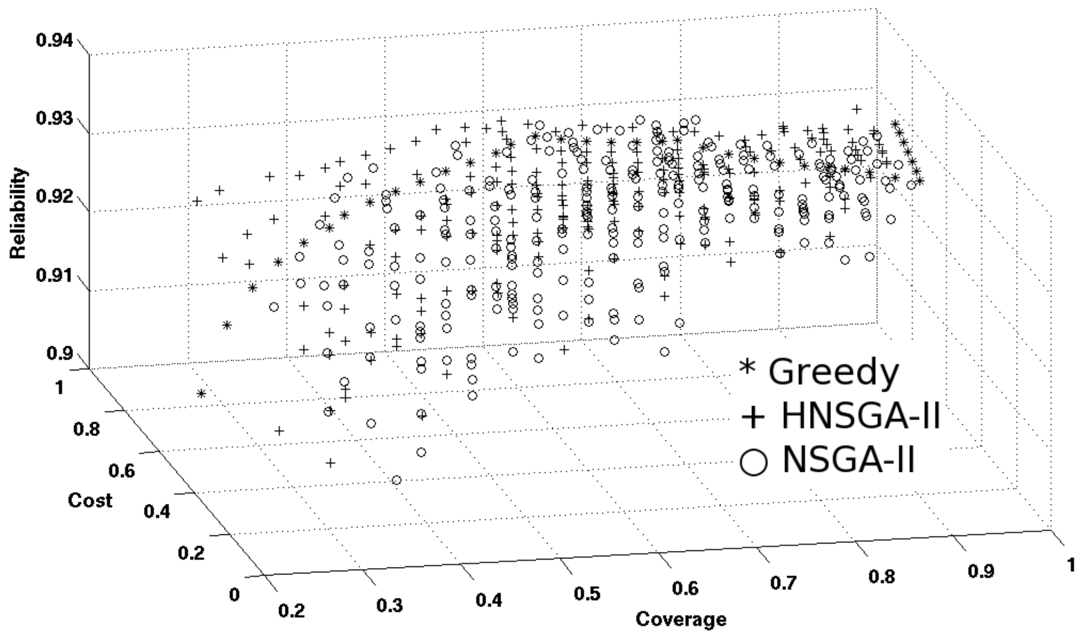


Figure 6.10: 3 objective optimisation for CS2 with per-use payment plan

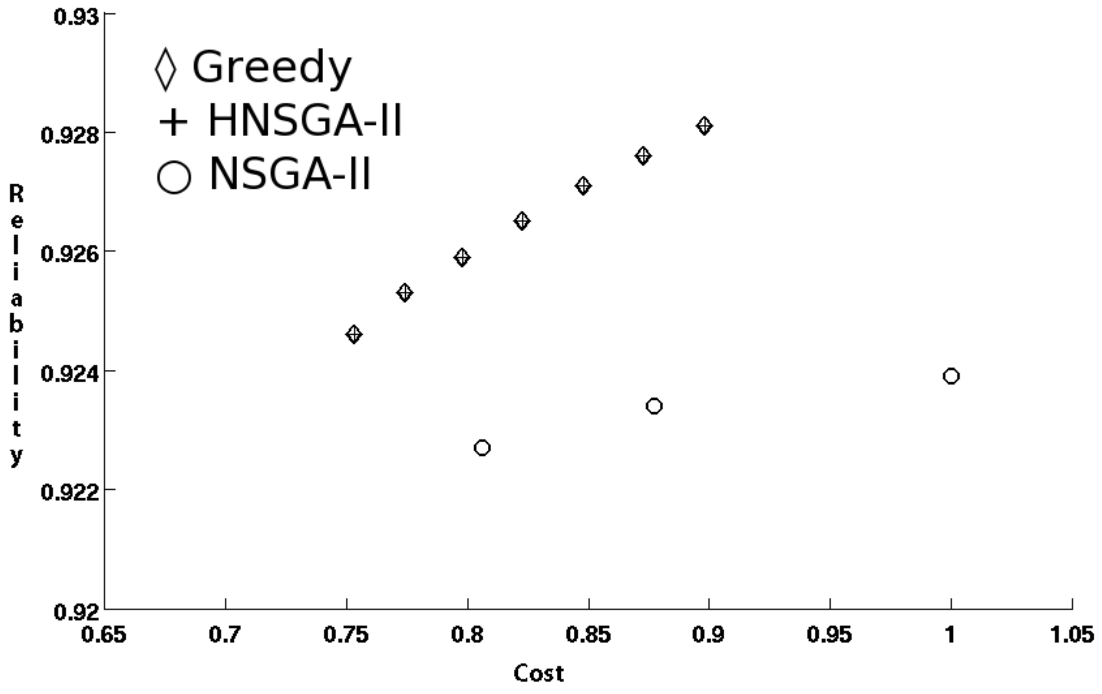


Figure 6.11: Projected view of Figure 6.10 focusing on the solutions with 100% coverage score.

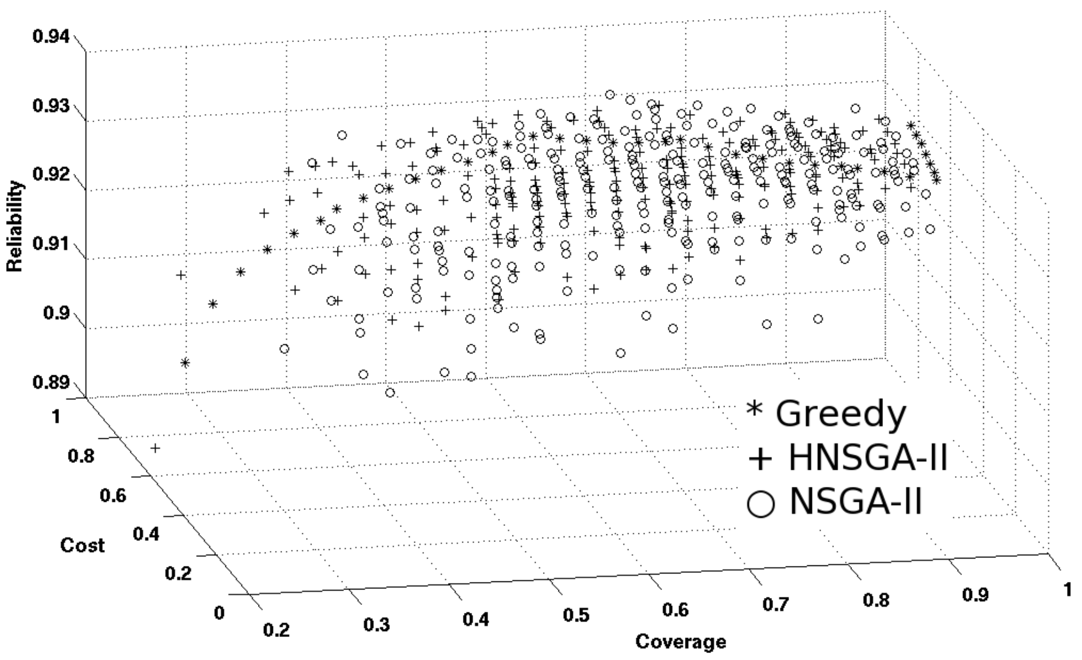


Figure 6.12: 3 objective optimisation for CS2 with contract-based payment plan

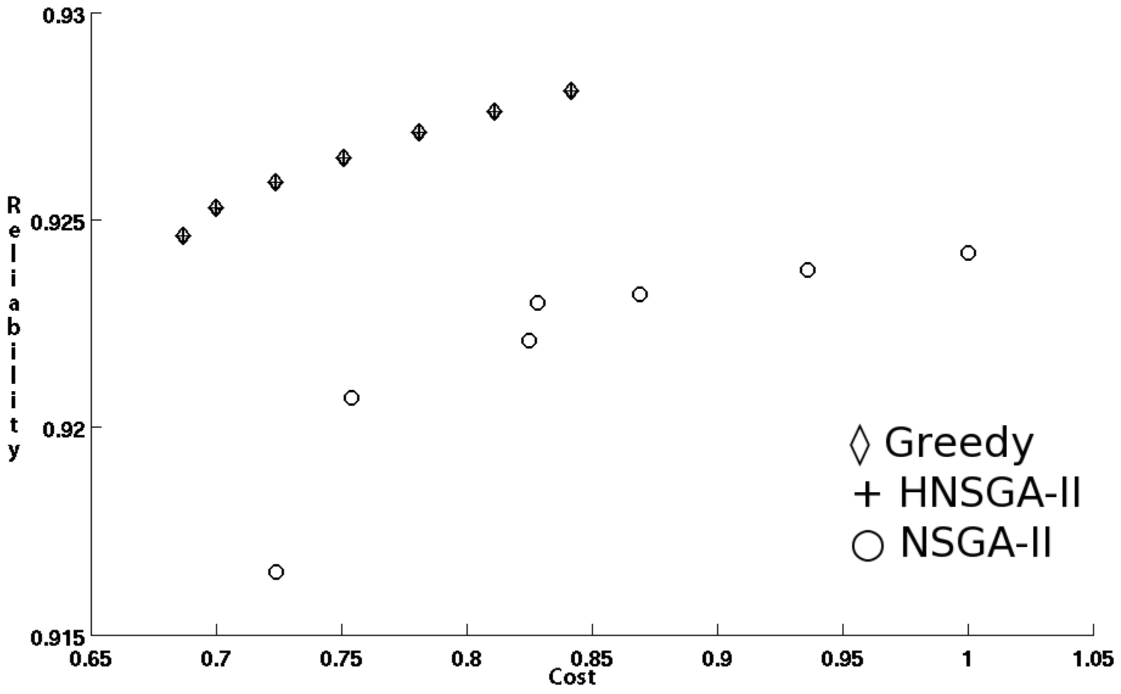


Figure 6.13: Projected view of Figure 6.12 focusing around the solutions with 100% coverage score.

The results revealed that the difference between the generated pareto fronts are not as high as presented in Table 6.12. Before the analysis, we expected the results to have a direct relation with the number of dominant solutions discovered. Our assumption was that the higher the number of dominant solutions, the higher the distance between the pareto fronts must be. The results from the analysis did not validate this assumption. However, the results provide evidence for the fact that for problems in which NSGA-II outperforms greedy, NSGA-II discovers solutions further from HNSGA-II. This can be observed in Figure 6.6 and 6.8 (a closer look of the high coverage areas of these two figures are depicted in Figure 6.7 and 6.9), where NSGA-II discovered solutions (especially around 100% coverage) that are far from the solutions discovered by HNSGA-II.

Experiment	Contract		Per Use	
	Average	σ	Average	σ
CS1 (2 objectives)	0.6999	0.71219	0.0353	0.05036
CS2 (2 objectives)	5.4224	1.83578	0.9128	0.24409
CS1 (3 objectives)	4.5524	2.28092	0.07	0.08572
CS2 (3 objectives)	1.521	0.42198	0.4014	0.11026

Table 6.12: The average distances between the pareto fronts generated by both algorithms. Both algorithms are run 10 times and the average Euclidean distance between pareto fronts are measured using the distance metric Υ discussed in the previous chapter.

The distances (between discovered fronts) depicted in figures mentioned here might not accurately represent the actual distance between the solutions due to cost values being normalised. The cost values (for both per-use and contract-based) are normalised in order to fit all discovered fronts into a single graph. However, the distances provided in tables are based on the calculations with the actual cost values and reflect the real distances between fronts.

The results provide evidence for the effectiveness of HSNGA-II over both NSGA-II and greedy algorithm. For example, results from Figure 6.11 and 6.13 clearly indicate that HNSGA-II can match greedy's performance for problems in which greedy discovers a pareto front close to optimum front and can outperform NSGA-II. On the other hand, results from Figure 6.7 and 6.9 indicate that for problems in which greedy can outperform NSGA-II, HNSGA-II also can outperform greedy by matching NSGA-II's performance.

6.4.2 Answers to Research Questions

The results from our experiments provide evidence for the effectiveness of our approach in testing cost reduction using multi-objective algorithms and answered **RQ1**. For both case studies, our approach achieved high reduction rates, with up to 84% reduction in the size of test suite and up to 99% reduction in the testing cost (for both payment plans) while retaining the coverage of the initial test suite.

With regards to **RQ2**, the results provide evidence for the fact that invalid inputs have a significant possibility of generating a false positive which might increase the testing cost in ScST. The observed false positive generation rates during the experiments were high (varying between 27% to 68%) for all the services we analysed except for one where the rate was 0.0014. The results also suggest that an invalid input is more likely to cause a false positive than the possibility of a realistic input causing an erroneous output in ScST.

As for **RQ3**, the results provide evidence for the fact that HNSGA-II performance is highly related to the performance of greedy algorithm. Since HNSGA-II executes the same NSGA-II algorithm for problems in which NSGA-II outperforms greedy (such as CS1), it performs similarly to NSGA-II. However, for the problems in which greedy outperforms NSGA-II, we observe the real benefit of HNSGA-II. The evidence for this claim came from the results of CS2, for both payment plans greedy discovered a very good set of solutions and HNSGA-II could not discover better results and resulted with the same pareto front. The advantage of using HNSGA-II is that it can match the performance of the best performing algorithm (out of the two) in any scenario, while outperforming the other one.

6.4.3 Threats to Validity

In the experiments, the most important external threat which limits the general applicability of the proposed approach is the test subjects representing real-world scenarios. This concern is especially important for synthetic cases studies. Although the ScS used in our experiments are synthetic examples, they are implemented merely to orchestrate the selected existing web services. Thus, these synthetic services definitely represent one of the possible orchestrations of these aforementioned services.

The other threat is the concept of test input reliability which is based on the reliability of the services used in generating it. The concept of service reliability, its formulation and threats regarding it were discussed in detail in Chapter 5.

The internal threats which might have affected the results obtained are:

1. *Coverage measurement*: We used a manual instrumentation method to measure the coverage scores of test cases. However, the correctness of the instrumentation was verified using the coverage measurement plug-in provided by Netbeans IDE.

2. *Service reliability measurements*: The most important factor that might affect the obtained reliability scores is our use of USPS as the ground truth for ZIP code validation. However, we believe that USPS must use one of the most up-to-date ZIP code databases since it is one of the government bodies that is responsible for address standardisation in the U.S.

3. *Price measurements*: Actual invocation prices of some of the services used in the experiments were not available to the authors during the experiments. As a result, the authors are forced to assign the invocation prices obtained for the experiments presented in Chapter 5 to these services. Despite not being the actual prices of these services, we believe they can serve a basis for a realistic scenario since they are real-world prices.

6.5 Conclusion

In this chapter, we introduced a solution aimed at reducing the runtime testing cost in ScST by using multi-objective optimisation, and presented an experimental study that investigated the relative effectiveness of proposed approach. We also investigated the HNSGA-II algorithm and compared its effectiveness in ScST against the other two well known algorithms (NSGA-II and greedy) used in test suite minimisation. In this part of our research agenda, we focused on the cost of runtime testing, branch coverage of the test suite and test suite reliability as our three primary objectives. The results provide evidence for the applicability and the effectiveness of the approach to ScS. Our approach achieved high reduction rates in both case studies with all possible payment plans without reducing the coverage of the test suite. The results also affirm the benefits of using HNSHA-II over NSGA-II especially for certain problems.

Chapter 7

Conclusion

7.1 Summary of Achievements

The two main goals of this thesis were automating the generation of realistic test inputs and reducing the cost of testing in ScST. Cost reduction was addressed at two different levels: test data generation level and runtime level. The details of each goal are as follows:

1. To automate generation of realistic test inputs using ScTDG, which is achieved through the exploitation of the existing web services.

2. To reduce cost at test data generation level through the multi-objective formulation of the objectives at this level. This enables the tester to select low cost test cases and make trade-offs in test data source selection based on different QoS characteristics. To provide further cost reduction by minimising the human-oracle cost by enabling testers to select and use more reliable data sources.

3. To reduce testing cost at runtime level by adapting traditional multi-objective test suite minimisation to ScST by formulating the ScST concerns.

7.1.1 Automated Realistic Test Input Generation

Automated test data generation is one of the major goals of automated software testing. Many different techniques such as random generation and search-based techniques are used for automating this process. These techniques are effective in generating inputs for systems such as embedded systems and control systems. However, their effectiveness is very limited when it comes to generating data for other systems, such as online systems. This limitation is caused by the fact that many online systems require test inputs that are not only correctly formed according to structural rules, but also contain semantics that ties them to real-world entities. The challenge in automation of these inputs is posed by the latter requirement.

Test data generation for ScS was performed using the available service specifications such as WSDL and OWL-S. Existing WSDL-based test data generation approaches [16, 20, 29, 130, 185, 201, 235, 295] combine XML data information and domain knowledge to generate valid and 'more realistic' test inputs. Even though these approaches can generate structurally valid test inputs, they cannot generate semantic realism. Ontology based approaches [17, 77, 326, 338] also fail in the aspect of semantic realism.

With the introduction of services, SOA and concepts such as Data as a Service, data is becoming easily accessible to consumers [91]. SOA enables the automation of finding and accessing the data source. This thesis considered existing services as sources of realistic test inputs and proposes their use for the automation of realistic test data generation.

This thesis proposed a new technique called ScDTG. In ScTDG, compositions of existing services which can provide the required realistic inputs are discovered and invoked in an automated fashion. The automation aspect of ScTDG helps reduce dependence on existing data and the need for tester input. The technique also brings benefits such as tester specified test data generation and reduction in human-oracle cost. The empirical study provides evidence for the feasibility of the technique and its effectiveness.

7.1.2 Multi-objective Data Source Selection

One of the factors that contribute to the overall testing cost is the cost of test data generation. This cost is often associated with the effort it takes in manual test data generation and/or the time it takes to generate the test data in automated generation [162]. However, when the tester uses ScTDG, the cost of manual effort is replaced by the monetary cost of invoking services. There are also other concerns in ScTDG, such as reliability of test data (based on its source), response time of services and availability of services, that might affect the test data generation process. Consider a situation where the tester has to use services with high response time which determines the test data generation time. In this case, he/she might prefer to use the services with low response time in order to reduce the total test data generation time.

As with many real-world situations, it is safe to assume that the tester will end up dealing with multiple objectives in ScTDG. Thus, he/she will want to be able to select and use services based on multiple criteria. This thesis formulates service selection problem in ScTDG as a multi-objective optimisation problem using the mentioned objectives: cost, reliability, response time and availability. The tester also benefits from the pareto-optimality of the approach which enables him to observe the trade-offs between objectives and to make a more informed decision based on his preferences. The empirical study provides evidence for the ability of the approach to discover solution sets very near optimal pareto fronts and the benefits it provides even in problems with small solution sets, such as reduced execution time.

7.1.3 Cost-aware Test Suite Minimisation

According to the literature, one of the limitations in ScST is the cost that is associated with invoking services. Existing approaches that address this problem offer solutions such as simulated testing and the use of stub/mock services. However, these approaches do not eliminate the need for runtime testing where ScS tested with real service invocations.

The literature also suggests that one of the well-known ways of reducing the cost of testing is test suite minimisation [374]. Even though the cost (of a test case) is often regarded as the execution time in existing work, this thesis replaces this notion with the invocation cost of services. The thesis introduces the concept of cost-aware test suite minimisation in ScST. The proposed approach is a multi-objective formulation of the problem which also considers other concerns such as test suite reliability and coverage. The approach enables the tester to make trade-offs between objectives due to its pareto-optimality. The empirical study provides evidence for the ability of the approaches to reduce the cost of testing while retaining branch coverage. The results also provide evidence to its ability to discover highly reliable test suites without increasing the testing cost.

7.2 Future Work

This section describes the future research direction where exactly the solutions described in this thesis can be extended.

7.2.1 ATAM as a Test Data Generator

An automated realistic test data generator is a goal not yet accomplished. The existing tools require too much manual effort and remains highly dependent on existing accessible test data sources. Unfortunately, these approaches might not be sufficient for testing ScS. What is required is a tool that is capable of generating realistic data and that can also be used in professional testing environments. ATAM was developed to fill this gap. Although ATAM helped automate realistic test data generation in experimental analysis, it does not fulfil the latter requirement.

As mentioned in Chapter 4, ATAM is designed as a prototype tool and does not support all possible service interactions and web service specifications. There are three major improvements needed to be implemented in order to make ATAM a complete testing tool: The first needed improvement is the redesign of the UI in order to make ATAM capable of representing services with multiple inputs and services that provide inputs for multiple services. The new design will also need to incorporate the missing functionality in the existing version such as dynamic constraint window building based on the structure of the data ontology and a more expressive data window for service groups. The second important addition is the support of all service specifications such as WSDL-S and especially Web Application Description Language (WADL), which is commonly used for describing RESTful services. The increase in the acceptance of REST among data services makes this improvement a top priority. The third improvement is the addition of full support for RDF. This addition combined with its OWL support will make ATAM compatible with most existing ontologies. The aforementioned additions are the first step in making ATAM a complete test data generator.

7.2.2 Pareto-optimal Service Composition Problem

Selection of optimal set of services is one of the major goals in automated service composition. Almost all of the existing approaches consider global composition constrains as hard goals that needs to be fulfilled. However, they are divided into two main categories when it comes to the level where optimisation is applied [10]. The first group applies optimisation at the global level, whereas the other group suggest optimisation at the local level. However, we believe that these approaches are not well suited to extreme changes in the QoS where a solution that fulfils the global requirements cannot be found. Ardagna and Pernici [10] address this issue by integrating the concept of QoS negotiation into the composition process. Unfortunately, the negotiation process cannot always guarantee a solution.

In the human decision process, it is important to see trade-offs between different solutions [103]. In order to make a more informed decision in multiple-criteria problems, humans often look at the improvements and decrements and assess their importance based on their situation. Unfortunately, at present the “optimal solution” in service composition is only based on the calculation of a few given QoS constraints. Existing work expects that the given constraints are good for any situation and expect the developer to specify these constraints. However, it is safe to assume that in an ideal composition process, the developer might expect to see the behaviour of the systems in situations where a composition fulfilling the given constraints cannot be found and specify some actions for those cases.

In order to address this problem, we propose the integration of human decisions (in the form of an oracle) from a range of possible scenarios into service composition problem especially targeting extreme situations. Decision oracles could be generated using the developer's decisions in various situations. The decisions will be collected from the training session where the developer is asked to choose the composition of their choice from the pareto-optimal solutions of different extreme QoS scenarios. The QoS scenarios used in training can be generated from existing trends in order to make the scenarios more realistic. This approach might also increase the developer's confidence by defining the expected composition in extreme situations. The proposed approach will be an extension of the work presented in Chapter 5 as it will benefit from its experiences and use its service selection framework.

7.2.3 Need for Test Suite Prioritisation

Selecting the execution order of the test cases can be as important as selecting the right test cases in many testing scenarios. Ordering is especially important if the tester is unsure if he/she will be able to execute all test cases in the time allocated for testing. As a result, the tester will want to run the most important/effective test cases as early as possible in order to get maximum benefit. Test case prioritisation seeks to find the ideal ordering of the test cases in order to maximise testing benefit if the process is stopped before the completion [374].

This scenario might occur in ScST where a developer is using a contract-based payment plan for service invocations. In this case, the testing might need to end after a certain number of invocations or a certain duration in order to avoid extra charges. Even though the test suite minimisation technique introduced in this thesis can help reduce runtime testing cost and help the developer to choose which test cases to run, there is still a need for the application of test suite prioritisation in ScST.

Hou et al [139] addressed some of these concerns in their multi-objective prioritisation approach, however, unaddressed concerns still remain. We believe their approach does not address two main concerns: assurance for the execution time of the selected test cases will not be more than the given time slot and execution of the test cases that cover the untested/selected parts of the SUT as early as possible. A new solution targeting these concerns might allow the developer to release their service earlier and it might be beneficial in projects with fast development cycles.

7.3 Closing Remarks

This thesis aimed at providing solutions to three important problems in ScST and introduced three novel solutions addressing these problems. The evidence acquired from the experimental analysis supports the hypothesis of using existing web services to automate realistic test data generation and using multi-objective search algorithms to reduce testing cost. The experimental results also provided evidence for the effectiveness of the proposed solutions compared to the state-of-the-art and the feasibility of their application to real world scenarios. The thesis also provided a roadmap to ScST researchers. The background information (in Chapter 2) and the analysis of the trends in ScST (in Chapter 3) might be useful especially to researchers who are new to the ScST domain.

References

[1] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, “A systematic review of the application and empirical investigation of search-based test case generation,” *IEEE Transactions on Software Engineering*, vol. 36, pp. 742–762, November 2010.

[2] Alive Matchmaker. Accessed: 27.07.2012. [Online]. Available: http://wiki.bath.ac.uk/display/alivedocs/

[3] N. Alshahwan and M. Harman, “Automated session data repair for web application regression testing.” in *ICST ’08: Proceedings of the 1st IEEE International Conference on Software Testing, Verification and Validation.* Lillehammer, Norway: IEEE, April 2008, pp. 298–307.

[4] N. Alshahwan and M. Harman, “Automated web application testing using search based software engineering,” in *ASE ’11: Proceedings of 26th IEEE/ACM International Conference On Automated Software Engineering*, Lawrence,KS,USA, November 2011, pp. 3–12.

[5] C. Andrés, M. Cambronero, and M. Núñez, “Passive testing of web services,” in *Web Services and Formal Methods*, ser. Lecture Notes in Computer Science, M. Bravetti and T. Bultan, Eds. Berlin / Heidelberg: Springer-Verlag, 2011, vol. 6551, pp. 56–70.

[6] Apache Web Services Project - Apache Axis. Accessed: 27.07.2012. [Online]. Available: http://ws.apache.org/axis/

[7] B. S. Apilli, “Fault-based combinatorial testing of web services,” in *OOPSLA ’09: Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. Orlando, Florida, USA: ACM, October 2009, pp. 731–732.

[8] AppLabs. (2007, May) Web services testing a primer. Accessed: 27.03.2012. [Online]. Available: http://www.docstoc.com/docs/4248976/Web-Services-Testing-A-Primer

[9] Apple iTunes. Accessed: 27.03.2012. [Online]. Available: http://www.apple.com/itunes/

[10] D. Ardagna and B. Pernici, “Adaptive service composition in flexible processes,” *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, Jun. 2007.

[11] A. Askarunisa, A. M. Abirami, and S. MadhanMohan, “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, pp. 1–7.

[12] A. Askarunisa, K. Punitha, and A. Askarunisa, “Black box test case prioritization techniques for semantic based composite web services using OWL-S,” in *ICRTIT 2011: Proceeding of the International Conference on Recent Trends in Information Technology*. Chennai, India: IEEE, July 2011, pp. 1215–1220.

[13] B. Athira and P. Samuel, “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, pp. 438–443.

[14] C. Atkinson, F. Barth, D. Brenner, and M. Schumacher, “Testing web-services using test sheets,” in *ICSEA ’10: Proceedings of the 5th International Conference on Software Engineering Advances*. Nice, France: IEEE Computer Society, August 2010, pp. 429–434.

[15] C. Atkinson, D. Brenner, G. Falcone, and M. Juhasz, "Specifying high-assurance services," *Computer*, vol. 41, no. 8, pp. 64–71, Aug. 2008.

[16] X. Bai, W. Dong, W. T. Tsai, and Y. Chen, "WSDL-based automatic test case generation for web services testing," in *SOSE '05: Proceedings of the 2006 IEEE International Workshop on Service-Oriented System Engineering*. Beijing, China: IEEE Computer Society, Oct. 2005, pp. 207–212.

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

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

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

[20] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "WS-TAXI: A WSDL-based testing tool for web services," in *ICST '09: Proceedings of the 2nd IEEE International Conference on Software Testing, Verification and Validation*. Denver, Colorado, USA: IEEE Computer Society, 2009, pp. 326–335.

[21] G. Bechara. (2009, March) What is a reusable service? Accessed: 27.03.2012. [Online]. Available: http://www.oracle.com/technetwork/articles/bechara-reusable-service-087796.html

[22] F. Belli, A. Endo, M. Linschulte, and A. Simao, "Model-based testing of web service compositions," in *SOSE '11: Proceedings of the 2011 IEEE Symposium on Service-Oriented System Engineering*. Irvine, CA, USA: IEEE Computer Society, December 2011, pp. 13–24.

[23] F. Belli and M. Linschulte, "Event-driven modeling and testing of real-time web services," *Service Oriented Computing and Applications*, vol. 4, pp. 3–15, 2010.

[24] C. Benedetto. (2006, September) SOA and integration testing: The end-to-end view. Accessed: 27.03.2012. [Online]. Available: http://soa.sys-con.com/node/275057

[25] A. Benharref, R. Dssouli, M. Serhani, A. En-Nouaary, and R. Glitho, "New approach for EFSM-based passive testing of web services," in *Testing of Software and Communicating Systems*, ser. Lecture Notes in Computer Science, A. Petrenko, M. Veanes, J. Tretmans, and W. Grieskamp, Eds. Berlin / Heidelberg: Springer, 2007, vol. 4581, pp. 13–27.

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

[27] A. Bertolino, G. De Angelis, L. Frantzen, and A. Polini, "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), ser. Lecture Notes in Computer Science, no. 5047. Tokyo, Japan: Springer, 2008, pp. 266–282.

[28] A. Bertolino, G. De Angelis, and A. Polini, "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, pp. 137–140.

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

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

[31] F. Bessayah, A. Cavalli, W. Maja, E. Martins, and A. Valenti, "A fault injection tool for testing web services composition," in *Testing – Practice and Research Techniques*, ser. Lecture Notes in Computer Science, L. Bottaci and G. Fraser, Eds. Berlin / Heidelberg: Springer, 2010, vol. 6303, pp. 137–146.

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

[33] A. Betin-Can, S. Hallé, and T. Bultan, "Modular verification of asynchronous service interactions using behavioral interfaces," *IEEE Transactions on Services Computing*, vol. PP, p. 1, November 2011.

[34] S. Bhiri, W. Gaaloul, M. Rouached, and M. Hauswirth, "Semantic web services for satisfying SOA requirements," in *Advances in Web Semantics I*, ser. Lecture

Notes in Computer Science, T. Dillon, E. Chang, R. Meersman, and K. Sycara, Eds. Berlin / Heidelberg: Springer, 2009, vol. 4891, pp. 374–395.

[35] Bing mobile. [Online]. Available: http://itunes.apple.com/us/app/bing/id345323231?mt=8

[36] F. Biscotti, Y. V. Natis, and M. Pezzini. (2012, October) Market trends: Platform as a service, worldwide, 2012-2016, 2h12 update. Gartner. Accessed: 20.11.2012. [Online]. Available: http://my.gartner.com/portal/server.pt?open=512&objID=202&&PageID=5553&mode=2&in_hi_userid=2&cached=true&resId=2188816&ref=AnalystProfile

[37] R. Blanco, J. García-Fanjul, and J. Tuya, "A first approach to test case generation for BPEL compositions of web services using scatter search," in *ICSTW '09: Proceedings of the 2nd IEEE International Conference on Software Testing, Verification, and Validation Workshops*. Denver, CO, USA: IEEE Computer Society, 2009, pp. 131–140.

[38] BLAST. Accessed: 27.03.2012. [Online]. Available: http://mtc.epfl.ch/software-tools/blast/

[39] J. Bloomberg. (2002, September) Web services testing: Beyond SOAP. Accessed: 27.03.2012. [Online]. Available: http://searchsoa.techtarget.com/news/article/0,289142,sid26_gci846941,00.html

[40] P. Boonyakulsrirung and T. Suwannasart, "A weak mutation testing framework for WS-BPEL," in *JCSSE 2011: Proceeding of the 8th International Joint Conference on Computer Science and Software Engineering*. Nakhon Pathom, Thailand: IEEE, May 2011, pp. 313–318.

[41] M. Bozkurt and M. Harman. (2008, August) Finding test data on the web. Presented at the 2008 Testing: Academic & Industrial Conference (TAIC PART 2008). [Online]. Available: http://www2008.taicpart.org/FastAbstracts/camera_ready/FastAbstract-11551.pdf

[42] M. Bozkurt and M. Harman, "Automatically generating realistic test input from web services," in *SOSE '11: Proceedings of the 2011 IEEE Symposium on Service-Oriented System Engineering*. Irvine, CA, USA: IEEE Computer Society, December 2011, pp. 13–24.

[43] M. Bozkurt and M. Harman, "Optimised realistic test input generation using web services," in *SSBSE 2012: Proceedings of the 4rd International Symposium on Search Based Software Engineering*, G. Fraser and J. Teixeira de Souza, Eds., vol. 7515. Riva Del Garda, Italy: Springer Berlin / Heidelberg, September 2012, pp. 105–120.

[44] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing & verification in service-oriented architecture: A survey," *Software Testing, Verification and Reliability (STVR)*, 2012, To Appear. [Online]. Available: http://dx.doi.org/10.1002/stvr.1470

[45] A. Bucchiarone, H. Melgratti, and F. Severoni, "Testing service composition," in *Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE07)*, Mar del Plata, Argentina, 2007.

[46] Calculator web service. Accessed: 27.07.2012. [Online]. Available: http://www.html2xml.nl/Services/Calculator/Version1/Calculator.asmx?WSDL

[47] M. Campanella, P. Chivalier, A. Sevasti, and N. Simar, *Quality of Service Definition*, Service Quality across Independently Managed Networks (SEQUIN), March 2001.

[48] G. Canfora and M. Di Penta, "Testing services and service-centric systems: challenges and opportunities," *IT Professional*, vol. 8, no. 2, pp. 10–17, March 2006.

[49] G. Canfora and M. Di Penta, "SOA: Testing and self-cheking," in *Proceedings of the International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, A. Bertolino and A. Polin, Eds., Palermo, Italy, June 2006, pp. 3–12. [Online]. Available: www.selab.isti.cnr.it/ws-mate/WS-MaTe_Proceedings.pdf

[50] T.-D. Cao, R. Castanet, P. Félix, and G. Morales, “Testing of web services: Tools and experiments,” in *APSCC 2011: IEEE Asia-Pacific Services Computing Conference*. Jeju, Korea: IEEE, December 2011, pp. 78–85.

[51] T.-D. Cao, P. Felix, R. Castanet, and I. Berrada, “Testing web services composition using the TGSE tool,” in *Proceedings of the 2009 Congress on Services - I*. Los Angeles, CA, USA: IEEE Computer Society, July 2009, pp. 187–194.

[52] T.-D. Cao, P. Felix, R. Castanet, and I. Berrada, “Online testing framework for web services,” in *ICST ’10: Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation*. Paris, France: IEEE Computer Society, April 2010, pp. 363–372.

[53] T.-D. Cao, T.-T. Phan-Quang, P. Flix, and R. Castanet, “Automated runtime verification for web services,” in *ICWS ’10: Proceedings of the 2010 IEEE International Conference on Web Services*. Miami,FL,USA: IEEE Computer Society, July 2010, pp. 76–82.

[54] G. B. L. Carlos A. Coello Coello and D. A. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition*, 2nd ed. Springer US, 2007.

[55] J. Carroll and D. Long, *Theory of finite automata with an introduction to formal languages*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[56] G. Carrozza, A. Napolitano, N. Laranjeiro, and M. Viera, “Wsrtesting: Hands-on solution to improve web services robustness testing,” in *LADCW 2011: The 5th Latin-American Symposium on Dependable Computing Workshops*. Sao Jose dos Campos, Brazil: IEEE, April 2011, pp. 41–46.

[57] R. Casado, J. Tuya, and M. Younas, “An abstract transaction model for testing the web services transactions,” in *ICWS 2011: Proceedings of the 9th IEEE International Conference on Web Services*. Washington DC, USA: IEEE, July 2011, pp. 730–731.

[58] R. Casado, J. Tuya, and C. Godart, “Dependency-based criteria for testing web services transactional workflows,” in *NWeSP 2011: Proceedings of the 7th International Conference on Next Generation Web Services Practices*. Salamanca, Spain: IEEE, October 2011, pp. 74–79.

[59] R. Casado, J. Tuya, and M. Younas, “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, July 2010, pp. 337–340.

[60] R. Casado, J. Tuya, and M. Younas, “A framework to test advanced web services transactions,” in *ICST 2011: Proceeding of the 4th International Conference on Software Testing, Verification and Validation*. Berlin, Germany: IEE, March 2011, pp. 443–446.

[61] R. Casado, J. Tuya, and M. Younas, “Testing the reliability of web services transactions in cooperative applications,” in *SAC ’12: Proceedings of the 27th Annual ACM Symposium on Applied Computing*. Trento, Italy: ACM, 2012, pp. 743–748.

[62] A. Cavalli, C. Gervy, and S. Prokopenko, “New approaches for passive testing using an extended finite state machine specification,” *Information and Software Technology*, vol. 45, no. 12, pp. 837–852, 2003.

[63] A. R. Cavalli, T.-D. Cao, W. Mallouli, E. Martins, A. Sadovykh, S. Salva, and F. Zadi, “Webmov: A dedicated framework for the modelling and testing of web services composition,” in *ICWS ’10: Proceedings of the 2010 IEEE International Conference on Web Services*. Paris, France: IEEE Computer Society, April 2010, pp. 377–384.

[64] CBDI Forum. (2002) Accessed: 27.03.2012. [Online]. Available: http://www.cbdiforum.com/

[65] S. K. Chakrabarti and P. Kumar, "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, pp. 302–308.

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

[67] D. Chandramohan, S. Jayakumar, S. Khapre, and M. Kishore, "DWSE-simulator for distributed web service environment," in *ICRTIT 2011: Proceedings of the 2011 International Conference on Recent Trends in Information Technology*. Chennai, India: IEEE, June 2011, pp. 1203–1208.

[68] S. Chandrasekaran, J. Miller, G. Silver, I. B. Arpinar, and A. Sheth, "Composition, performance analysis and simulation of web services," *Electronic Markets: The International Journal of Electronic Commerce and Business Media (EM)*, vol. 13, no. 2, p. 120132., Nuje 2003.

[69] A. Chaturvedi, "Reducing cost in regression testing of web service," in *CONSEG 2012: Proceedigs of the 6th International Conference on Software Engineering*. Indore, India: IEEE, September 2012, pp. 1–9.

[70] H. Chen, H. Jin, F. Mao, and H. Wu, "Q-GSM: QoS oriented grid service management," in *Proceedings of the 7th Asia-Pacific web conference on Web Technologies Research and Development (APWeb 2005)*. Shanghai, China: SpringerLink, March 2005, pp. 1041–1044.

[71] L. Chen, Z. Wang, L. Xu, H. Lu, and B. Xu, "Test case prioritization for web service regression testing," in *SOSE '10: Proceedings of the 2010 IEEE International Symposium on ServiceOriented System Engineering*. Loughborough, United Kingdom: IEEE Computer Society, June 2010, pp. 173–178.

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

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

[74] K. Conroy, M. Grechanik, M. Hellige, E. Liongosari, and Q. Xie, "Automatic test generation from GUI applications for testing web services," in *ICSM '07: Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM 2007).* Paris, France: IEEE Computer Society, Oct. 2007, pp. 345–354.

[75] CPNTOOLS. Accessed: 27.03.2012. [Online]. Available: http://wiki.daimi.au.dk/cpntools/cpntools.wiki

[76] U. Dahan. (2006, ,July) Autonomous services and enterprise entity aggregation. [Online]. MSDN. Accessed: 27.03.2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb245672.aspx

[77] G. Dai, X. Bai, Y. Wang, and F. Dai, "Contract-based testing for web services," in *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference*, vol. 1. Beijing, China: IEEE Computer Society, July 2007, pp. 517–526.

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

[79] L. F. J. de Almeida and S. R. Vergilio, "Exploring perturbation based testing for web services," in *ICWS '06: Proceedings of the 2006 IEEE International*

Conference on Web Services. Chicago, IL, USA: IEEE Computer Society, 2006, pp. 717–726.

[80] F. De Angelis, A. Polini, and G. De Angelis, “A counter-example testing approach for orchestrated services,” in *ICST ’10: Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation.* Paris, France: IEEE Computer Society, 2010, pp. 373–382.

[81] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Trans. Evolutionary Computation*, pp. 182–197, 2002.

[82] M. Di Penta, M. Bruno, G. Esposito, V. Mazza, and G. Canfora, “Web services regression testing,” in *Test and Analysis of Web Services*, L. Baresi and E. Di Nitto, Eds. Berlin / Heidelberg: Springer-Verlag, 2007, pp. 205–234.

[83] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno, “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, pp. 1090–1097.

[84] I. Di Pietro, F. Pagliarecci, and L. Spalazzi, “Model checking semantically annotated services,” *IEEE Transactions on Software Engineering*, vol. 38, pp. 592 – 608, January 2012.

[85] Disjunctive Normal Form. Accessed: 27.03.2012. [Online]. Available: http://mathworld.wolfram.com/DisjunctiveNormalForm.html

[86] J. J. Domínguez-Jiménez, A. Estero-Botaro, A. García-Domínguez, and I. Medina-Bulo, “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, pp. 97–106.

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

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

[89] M. Driss, Y. Jamoussi, and H. H. B. Ghezala, “QoS testing of service-based applications,” in *Proceedings of the 3rd IEEE International Design and Test Workshop (IDT ’08).* Monastir, Tunisia: IEEE, December 2008, pp. 45–50.

[90] P. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk.* Upper Saddle River, NJ, USA: Addison-Wesley Professional, 2007.

[91] J. Dyche. (2007, July) Data-as-a-service, explained and defined. Accessed: 24.10.2012. [Online]. Available: http://searchdatamanagement.techtarget.com/answer/Data-as-a-service-explained-and-defined

[92] ECJ 20. [Online]. Available: http://cs.gmu.edu/~eclab/projects/ecj/

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

[94] N. El Ioini, “Web services open test suites,” in *Proceedings of the 2011 IEEE World Congress on Services (SERVICES).* Washington, DC, USA: IEEE, July 2011, pp. 77–80.

[95] N. El Ioini and A. Sillitti, “Open web services testing,” in *Proceedings of the 2011 IEEE World Congress on Services (SERVICES)*. Washington, DC, USA: IEEE, July 2011, pp. 130–136.

[96] M. M. Eler, M. E. Delamaro, J. C. Maldonado, and P. C. Masiero, “Built-in structural testing of web services,” in *SBES ’10: Proceedings of the 2010 Brazilian Symposium on Software Engineering*. Salvador, Brasil: IEEE Computer Society, October 2010, pp. 70–79.

[97] A. T. Endo, M. Linschulte, A. S. Simão, and S. R. S. Souza, “Event- and coverage-based testing of web services,” in *Proceedings of the 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement Companion*, ser. SSIRI-C ’10. Singapore, Singapore: IEEE Computer Society, April 2010, pp. 62–69.

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

[99] A. Endo and A. Simao, “Model-based testing of service-oriented applications via state models,” in *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC)*. Washington, DC: IEEE, July 2011, pp. 432–439.

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

[101] J. P. Escobedo, C. Gaston, P. Le Gall, and A. Cavalli, “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, September 2010, pp. 257–267.

[102] ETSI ES 201 873-1. The Testing and Test Control Notation version 3, Part1: TTCN-3 Core notation, V2.1.1. Accessed: 27.03.2012. [Online]. Available: http://www.ttcn-3.org/StandardSuite.htm

[103] M. Farina and P. Amato, "A fuzzy definition of "optimality" for many-criteria optimization problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, pp. 315–326, 2004.

[104] K. Farj, Y. Chen, and N. Speirs, "A fault injection method for testing dependable web service systems," in *ISORC 2012: Proceedings of the 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. Guangdong, China: IEEE, April 2012, pp. 47–55.

[105] FedEx Rate Calculator. Accessed: 27.07.2012. [Online]. Available: https://www.fedex.com/ratefinder/home?cc=US&language=en&locId=express

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

[107] M. Felderer, P. Zech, F. Fiedler, and R. Breu, "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, August 2010, pp. 108–113.

[108] FIT: Framework for Integrated Test. Accessed: 27.03.2012. [Online]. Available: http://fit.c2.com/

[109] Flurry.com. [Online]. Available: http://blog.flurry.com/?BBPage=1

[110] J. Fox and J. Borenstein. (2003, March) Semantic discovery for web services. SOA World Magazine. Accessed: 27.03.2012. [Online]. Available: http://soa.sys-con.com/node/39718

[111] L. Frantzen, M. Las Nieves Huerta, Z. G. Kiss, and T. Wallet, "On-the-fly model-based testing of web services with jambition," in *Web Services and Formal Methods*, ser. Lecture Notes in Computer Science, R. Bruni and K. Wolf, Eds. Berlin, Heidelberg: Springer-Verlag, 2009, vol. 5387, pp. 143–157.

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

[113] D. Fu and G. Chen, "A verification method for web services combination based on abstract and refinement technology," in *CECNet 2012: Proceedings of the 2nd International Conference on Consumer Electronics, Communications and Networks.* Yichang, China: IEEE, April 2012, pp. 119 – 122.

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

[115] H. Gao and Y. Li, "Generating quantitative test cases for probabilistic timed web service composition," in *Proceedings of the 2011 IEEE Asia-Pacific Services Computing Conference (APSCC).* Jeju, Korea: IEEE, December 2011, pp. 275–283.

[116] H. Gao, H. Miao, S. Chen, and J. Mei, "Probabilistic timed model checking for atomic web service," in *SERVICES 2011: Proceedings of the 2011 IEEE World Congress on Services.* Washington, DC, USA: IEEE, July 2011, pp. 459 – 466.

[117] J. García-Fanjul, C. de la Riva, and J. Tuya, "Generating test cases specifications for compositions of web services," in *Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, A. Bertolino and A. Polini, Eds., Palermo, Italy, June 2006, pp. 83–94. [Online]. Available: www.selab.isti.cnr.it/ws-mate/WS-MaTe_Proceedings.pdf

[118] M. Garriga, A. Flores, A. Cechich, and A. Zunino, "Testing-based process for service-oriented applications," in *Proceedings of the 30th International Conference of the Chilean Computer Science Society*. Curico, Chile: IEEE, November 2011, pp. 64–73.

[119] A. Gill, *Introduction to the theory of finite-state machines*. McGraw-Hill, 1962.

[120] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Addison-Wesley Professional, Jan. 1989.

[121] Google search. [Online]. Available: http://itunes.apple.com/us/app/google-search/id284815942?mt=8

[122] E. Grishikashvili, D. Reilly, N. Badr, and A. Taleb-Bendiab, "From component-based to service-based distributed applications assembly and management," in *EUROMICRO '03: Proceedings of the 29th EUROMICRO Conference*. Antalya, Turkey: IEEE Computer Society, Sept. 2003, pp. 99–106.

[123] J. Grundy, J. Hosking, L. Li, and N. Liu, "Performance engineering of service compositions," in *SOSE '06: Proceedings of the 2006 IEEE International workshop on Service-oriented software engineering*. Shanghai, China: ACM, 2006, pp. 26–32.

[124] Y. Gu and Y. Ge, "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, pp. 320–324.

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

[126] N. Guermouche and C. Godart, “Timed model checking based approach for web services analysis,” in *ICWS ’09: Proceedings of the 2009 IEEE International Conference on Web Services*. Los Angeles, CA, USA: IEEE Computer Society, July 2009, pp. 213–221.

[127] S. Hallé, “Model-based simulation of soapweb services from temporal logic specifications,” in *ICECCS 2011: proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems*. Las Vegas, NV, USA: IEEE, April 2011, pp. 95–104.

[128] S. Hallé, R. Villemaireand, O. Cherkaoui, and R. Deca, “A logical approach to data-aware automated sequence generation,” *Lecture Notes in Computer Science*, vol. 7050, pp. 192–216, 2012.

[129] L. Hamel, M. Graiet, M. Kmimech, M. T. Bhiri, and W. Gaaloul, “Verifying composite service transactional behavior with EVENT-B,” in *Proceeedings of the 2011 Seventh International Conference on Semantics, Knowledge and Grids*. Beijing, China: IEEE, October 2011, pp. 99 – 106.

[130] S. Hanna and M. Munro, “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, pp. 471–476.

[131] M. Harman, “Automated test data generation using search based software engineering,” in *AST ’07: Proceedings of the 2nd International Workshop on Automation of Software Test*. Minneapolis, MN, USA: IEEE Computer Society, May 2007, p. 2.

[132] M. Harman, “The current state and future of search based software engineering,” in *FOSE: Future of Software Engineering 2007 (FOSE ’07)*, Washington, DC, USA, May 2007, pp. 342–357.

[133] M. Harman, "Making the case for MORTO: Multi objective regression test optimization," in *Regression 2011*, Berlin, Germany, March 2011.

[134] M. Harman, F. Islam, T. Xie, and S. Wappler, "Automated test data generation for aspect-oriented programs," in *Proceedings of the 8th ACM international conference on Aspect-oriented software development*, ser. AOSD '09. New York, NY, USA: ACM, 2009, pp. 185–196.

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

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

[137] R. Heckel and L. Mariani, "Automatic conformance testing of web services," in *FASE 2005: Proceedings of the Fundamental Approaches to Software Engineering*. Edinburgh, Scotland: Springer, April 2005, pp. 34–48.

[138] S. S. Hou, L. Zhang, Q. Lan, H. Mei, and J. S. Sun, "Generating effective test sequences for BPEL testing," in *QSIC 2009: Proceedings of the 9th International Conference on Quality Software*. Jeju, Korea: IEEE Computer Society Press, August 2009, pp. 331–340.

[139] S.-S. Hou, L. Zhang, T. Xie, and J.-S. Sun, "Quota-constrained test-case prioritization for regression testing of service-centric systems," in *ICSM '08: Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM 2008)*. Beijing, China: IEEE, Oct. 2008, pp. 257–266.

[140] HP Service Test. Accessed: 27.03.2012. [Online]. Available: http://h71028.www7.hp.com/enterprise/cache/19054-0-0-225-121.html

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

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

[143] S.-Y. Hwang, W.-F. Hsieh, and C.-H. Lee, “Verifying web services in a choreography environment,” in *SOCA 2011: Proceedings of the 2011 IEEE International Conference on Service-Oriented Computing and Applications.* Irvine, CA, USA: IEEE, December 2011, pp. 1–4.

[144] N. Ibrahim and I. Khalil, “Verifying web services compositions using UPPAAL,” in *Proceedings of the 2012 International Conference on Computer Systems and Industrial Informatics (ICCSII).* Sharjah, UAE: IEEE, January 2012, pp. 1 – 5.

[145] S. Ilieva, I. Manova, and D. Petrova-Antonova, “Towards a methodology for testing of business processes,” in *Proceeding of the 2012 Federated Conference on Computer Science and Information Systems (FedCSIS).* Wrocaw, Poland: IEEE, September 2012, pp. 1315–1322.

[146] S. Ilieva, V. Pavlov, and I. Manova, “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, 2010, pp. 286–291.

[147] D. C. Ince, “The automatic generation of test data,” *The Computer Journal*, vol. 30, no. 1, pp. 63–69, 1987.

[148] International Data Corporation (IDC). Accessed: 27.03.2012. [Online]. Available: http://www.idc.com/

[149] A. Iskold. (2007, March) Web 3.0: When web sites become web services. Accessed: 27.03.2012. [Online]. Available: http://www.readwriteweb.com/archives/web_30_when_web_sites_become_web_services.php

[150] Java PathFinder. Accessed: 27.03.2012. [Online]. Available: http://javapathfinder.sourceforge.net/

[151] K. Jensen, "Coloured petri nets: Status and outlook," in *Applications and Theory of Petri Nets 2003*, ser. Lecture Notes in Computer Science, W. van der Aalst and E. Best, Eds. Berlin / Heidelberg: Springer, 2003, vol. 2679, pp. 1–2.

[152] JESS rule engine. Accessed: 27.03.2012. [Online]. Available: http://www.jessrules.com/jess/index.shtml

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

[154] M. S. Jokhio, G. Dobbie, and J. Sun, "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, December 2009, pp. 17–24.

[155] J. A. Jones, M. J. Harrold, and J. T. Stasko, "Visualization for fault localization," in *in Proceedings of ICSE 2001 Workshop on Software Visualization*, Toronto, Canada, May 2001, pp. 71–75.

[156] S. Jones, "Toward an acceptable definition of service [service-oriented architecture]," *IEEE Software*, vol. 22, no. 3, pp. 87–93, May-June 2005.

[157] L. Juszczyk and S. Dustdar, "Automating the generation of web service testbeds using aop," in *ECOWS 2011: Proceedings of the 9th IEEE European Conference on Web Services*. Lugano, Switzerland: IEEE, September 2011, pp. 143–150.

[158] L. Juszczyk and S. Dustdar, "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, pp. 195–202.

[159] L. Juszczyk and S. Dustdar, "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, July 2010, pp. 683–686.

[160] JXML2OWL Project. Accessed: 27.03.2012. [Online]. Available: http://jxml2owl.projects.semwebcentral.org/

[161] H. Kacem, W. Sellami, and A. Kacem, "A formal approach for the validation of web service orchestrations," in *WETICE 2012: IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Toulouse, France: IEEE, June 2012, pp. 42 – 47.

[162] C. Kaner, J. Bach, and B. Pettichord, *Lessons learned in software testing*. John Wiley & Sons, 2008.

[163] S. Karre, "Leveraging user-session data to support web application testing," *IEEE Trans. Softw. Eng.*, vol. 31, no. 3, pp. 187–202, 2005, member-Elbaum,, Sebastian and Member-Rothermel,, Gregg and Member-Fisher II,, Marc.

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

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

[166] K. Lakhotia, M. Harman, and P. McMinn, "A multi-objective approach to search-based test data generation," in *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*. London, UK: ACM, July 2007, pp. 1098–1105.

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

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

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

[170] N. Laranjeiro, R. Oliveira, and M. Vieira, "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, pp. 255–264.

[171] N. Laranjeiro, M. Vieira, and H. Madeira, "Improving web services robustness," in *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*. Los Angeles, CA, USA: IEEE Computer Society, July 2009, pp. 397–404.

[172] G. Laycock, "The theory and practice of specification based software testing," Ph.D. dissertation, University of Sheffield, 2003.

[173] K. Lee, J. Jeon, W. Lee, S.-H. Jeong, and S.-W. Park. (2003, November) QoS for web services: Requirements and possible approaches. Accessed: 27.03.2012. [Online]. Available: http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/

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

[175] B. Legeard, "BZ-Testing-Tools: Model-based test generator," in *The 18th IEEE International Conference on Automated Software Engineering (ASE 2003) - Demo Paper*, Montreal, Canada, Oct. 2003. [Online]. Available: http://www.ase-conferences.org/ase/past/ase2003/demos/BZ-Testing-Tools%20at%20ASE2003.pdf

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

[177] B. Li, D. Qiu, S. Ji, and D. Wang, "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, September 2010, pp. 1–10.

[178] L. Li and W. Chou, "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, July 2009, pp. 205–212.

[179] L. Li and W. Chou, "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, July 2009, pp. 179–186.

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

[181] Q. Li, J. Chen, Y. Zhan, and C. Mao, "Combinatorial mutation approach to web service vulnerability testing based on SOAP message mutations," in *ICEBE 2012: Proceedings of the IEEE 9th International Conference on e-Business Engineering*. Hangzhou, China: IEEE, September 2012, pp. 156–162.

[182] X. Li, J. Huai, X. Liu, J. Zeng, and Z. Huang, "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, 2009, pp. 163–170.

[183] X. Li, Y. Fan, Q. Sheng, and Z. Maamar, "A petri net approach to analyzing behavioral compatibility and similarity of web services," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 41, pp. 510 – 521, January 2011.

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

[185] Z. J. Li, J. Zhu, L.-J. Zhang, and N. Mitsumori, "Towards a practical and effective method for web services test case generation," in *Proceedings of the ICSE*

Workshop on Automation of Software Test (AST'09). Vancouver, Canada: IEEE Computer Society, May 2009, pp. 106–114.

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

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

[188] F. Lin, M. Ruth, and S. Tu, "Applying safe regression test selection techniques to java web services," in *NWESP '06: Proceedings of the 3rd International Conference on Next Generation Web Services Practices*. Seoul, South Korea: IEEE Computer Society, Sept. 2006, pp. 133–142.

[189] C.-H. Liu, S.-L. Chen, J. Y. Kuo, and T.-Y. Huang, "A flow graph-based test model for owl-s web services," in *Proceedings of the 2011 International Conference on Machine Learning and Cybernetics (ICMLC) (Volume:2)*. Guilin, China: IEEE, July 2011, pp. 897–902.

[190] F. Liu, F. yuan Ma, , M. lu Li, and L. peng Huang, "A framework for semantic grid service discovery," in *Web Information Systems - WISE 2004 Workshops*. Brisbane, Australia: SpringerLink, November 2004, pp. 3–10.

[191] H. Liu, Z. Li, J. Zhu, and H. Tan, "Business process regression testing," in *Proceedings of the 5th international conference on Service-Oriented Computing*, ser. ICSOC '07. Vienna, Austria: Springer-Verlag, 2007, pp. 157–168.

[192] H. Liu, Z. Li, J. Zhu, H. Tan, and H. Huang, "A unified test framework for continuous integration testing of SOA solutions," in *ICWS '09: Proceedings of*

the 2009 IEEE International Conference on Web Services. Los Angeles, CA, USA: IEEE Computer Society, July 2009, pp. 880–887.

[193] J. Liu, X. Lu, X. Feng, and J. Liu, "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, December 2010, pp. 161–168.

[194] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing interacting BPEL processes," in *BPM 2006: 4th International Conference on Business Process Management*, ser. Lecture Notes in Computer Science, S. Dustdar, J. Fiadeiro, and A. Sheth, Eds., vol. 4102. Vienna, Austria: Springer-Verlag, September 2006, p. 1732.

[195] N. Looker, M. Munro, B. Gwynne, and J. Xu, "An ontology-based approach for determining the dependability of service-oriented architectures," in *WORDS '05: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*. Sedona, AZ, USA: IEEE Computer Society, Feb. 2005, pp. 171–178.

[196] N. Looker, J. Xu, and M. Munro, "Determining the dependability of service-oriented architectures," *International Journal of Simulation and Process Modelling*, vol. 3, no. 26, pp. 88–97, Jul. 2007.

[197] B. Lublinsky. (2007, October) Use service-oriented decomposition to meet your architectural goals. [Online]. IBM. Accessed: 27.03.2012. [Online]. Available: http://www.ibm.com/developerworks/library/ar-soadecomp/

[198] X. Luo, L. Luo, and M. Zou, "An epistemic model checking approach for owl-s web services," in *Proceedings of the 2012 IEEE Symposium on Electrical & Electronics Engineering (EEESYM)*. Kuala Lumpur, Malaysia: IEEE, June 2012, pp. 694 – 697.

[199] X. Luo, F. Ping, and M.-H. Chen, "Clustering and tailoring user session data for testing web applications," in *ICST '09: Proceedings of the 2009 International Conference on Software Testing Verification and Validation*. Denver, Colorado, USA: IEEE Computer Society, April 2009, pp. 336–345.

[200] X. Luo, F. Ping, and M.-H. Chen, "Clustering and tailoring user session data for testing web applications," in *ICST '09: Proceedings of the 2nd IEEE International Conference on Software Testing, Verification and Validation*. Denver, CO, USA: IEEE, April 2009, pp. 336–345.

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

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

[203] A. Maalej, M. Krichen, and M. Jmaiel, "Conformance testing of ws-bpel compositions under various load conditions," in *COMPSAC 2012: Proceedings of the IEEE 36th Annual Computer Software and Applications Conference*. Izmir, Turkey: IEEE, July 2012, pp. 371–376.

[204] A. Maalej, M. Krichen, and M. Jmaiel, "Model-based conformance testing of ws-bpel compositions," in *COMPSACW 2012: Proceedings of the IEEE 36th Annual Computer Software and Applications Conference Workshops*. Izmir, Turkey: IEEE, July 2012, pp. 452–457.

[205] S. Mani, V. S. Sinha, S. Sinha, P. Dhoolia, D. Mukherjee, and S. Chakraborty, "Efficient testing of service-oriented applications using semantic service stubs," in *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web*

Services. Los Angeles, CA, USA: IEEE Computer Society, July 2009, pp. 197–204.

[206] MaramaMTE. Accessed: 27.03.2012. [Online]. Available: https://wiki.auckland.ac.nz/display/csidst/MaramaMTE

[207] E. Martin, S. Basu, and T. Xie, "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, pp. 647–654.

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

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

[210] P. McMinn, M. Shahbaz, and M. Stevenson, "Search-based test input generation for string data types using the results of web queries," in *ICST '12: Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation*. Montreal, Canada: IEEE, April 2012, p. 141150.

[211] P. McMinn, M. Stevenson, and M. Harman, "Reducing qualitative human oracle costs associated with automatically generated test data," in *STOV '10: 1st International Workshop on Software Test Output Validation, in conjuntion with ICST '10*. Trento, Italy: ACM, July 2010, pp. 1–4.

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

[213] L. Mei, W. K. Chan, T. H. Tse, and R. G. Merkel, "Tag-based techniques for black-box test case prioritization for service testing," in *Proceedings of the 9th International Conference on Quality Software (QSIC 2009).* Jeju, Korea: IEEE Computer Society Press, August 2009, pp. 21–30.

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

[215] L. Mei, W. K. Chan, and T. H. Tse, "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, 2009, pp. 151–160.

[216] L. Mei, K. Zhai, B. Jiang, and W. Chan, "Preemptive regression test scheduling strategies: A new testing approach to thriving on the volatile service environments," in *COMPSAC 2012: Proceedings of the IEEE 36th Annual Computer Software and Applications Conference*. Izmir, Turkey: IEEE, July 2012, pp. 72–81.

[217] L. Mei, Z. Zhang, W. K. Chan, and T. H. Tse, "Test case prioritization for regression testing of service-oriented business applications," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. Madrid, Spain: ACM, 2009, pp. 901–910.

[218] METEOR-S. Accessed: 27.03.2012. [Online]. Available: http://lsdis.cs.uga.edu/projects/meteor-s/

[219] B. Meyer, "Applying 'Design by Contract'," *Computer*, vol. 25, no. 10, pp. 40–51, Oct 1992.

[220] W. Miao and S. Liu, "A formal specification-based testing approach to accurate web service selection," in *Proceedings of the 2011 IEEE Asia-Pacific Services Computing Conference (APSCC).* Jeju, Korea: IEEE, December 2011, pp. 259–266.

[221] MINDSWAP: Maryland information and network dynamics lab semantic web agents project. OWL-S services. Accessed: 27.03.2012. [Online]. Available: http://www.mindswap.org/2004/owl-s/services.shtml

[222] G. Morales, S. Maag, and A. Cavalli, "Timed extended invariants for the passive testing of web services," in *ICWS '10: Proceedings of the 2010 IEEE International Conference on Web Services.* Miami,FL,USA: IEEE Computer Society, July 2010, pp. 76–82.

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

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

[225] S. Moser, A. Martens, K. Görlach, W. Amme, and A. Godlinski, "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, pp. 98–105.

[226] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr 1989.

[227] M. Narita, M. Shimamura, K. Iwasa, and T. Yamaguchi, "Interoperability verification for web service based robot communication platforms," in *ROBIO 2007: Proceedings of the 2007 IEEE International Conference on Robotics and*

Biomimetics. Sanya, China: IEEE Computer Society, Dec. 2007, pp. 1029–1034.

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

[229] C. Nguyen, A. Marchetto, and P. Tonella, “Test case prioritization for audit testing of evolving web services using information retrieval techniques,” in *ICWS 2011: Proceedings of the 9th International Conference on Web Services*. Washington, DC, USA: IEEE, July 2011, pp. 636–643.

[230] C. Nguyen, A. Marchetto, and P. Tonella, “Challenges in audit testing of web services,” in *Proceedings of the 4th International Conference on Software Testing, Verification and Validation Workshops*. Berlin Germany: IEEE, March 2011, pp. 103–106.

[231] C. Nguyen, A. Marchetto, and P. Tonella, “Change sensitivity based prioritization for audit testing of webservice compositions,” in *Proceedings of the 4th International Conference on Software Testing, Verification and Validation Workshops*. Berlin Germany: IEEE, March 2011, pp. 103–106.

[232] S. Noikajana and T. Suwannasart, “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, July 2009, pp. 115–123.

[233] NuSMV. Accessed: 27.03.2012. [Online]. Available: http://nusmv.irst.itc.it/

[234] OASIS. SOA-EERP business quality of service (bQoS). Accessed: 27.03.2012. [Online]. Available: http://docs.oasis-open.org/ns/soa-eerp/bqos/200903

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

[236] G. Oghabi, J. Bentahar, and A. Benharref, “On the verification of behavioral and probabilistic web services using transformation,” in *ICWS 2011: proceedings of the 9th International Conference on Web Services*. Washington, DC, USA: IEEE, July 2011, pp. 548 – 555.

[237] R. Oliveira, N. Laranjeiro, and M. Vieira, “A composed approach for automatic classification of web services robustness,” in *Proceedings of the 2011 IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE, July 2011, pp. 176–183.

[238] Oracle Application Testing Suite. Accessed: 27.03.2012. [Online]. Available: http://www.oracle.com/technetwork/oem/app-quality-mgmt/application-quality-management-092933.html

[239] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*, 1st ed. The MIT Press, July 1994.

[240] N. Oster and F. Saglietti, “Automatic test data generation by multi-objective optimisation,” in *Computer Safety, Reliability, and Security*, ser. Lecture Notes in Computer Science, J. Górski, Ed. Springer Berlin, 2006, vol. 4166, pp. 426–438.

[241] C. Ouyang, H. M. W. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. H. M. ter Hofstede, “WofBPEL: A tool for automated analysis of BPEL processes,” in *Service-Oriented Computing - ICSOC 2005*, B. Benatallah, F. Casati, and P. Traverso, Eds. Berlin / Heidelberg: Springer-Verlag, 2005, vol. 3826, pp. 484–489.

[242] OWL-S: Semantic Markup for Web Services. Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[243] Oxford English Dictionary 2nd. ed. (2010) Accessed: 27.03.2012. [Online]. Available: http://dictionary.oed.com/cgi/entry/50015226

[244] W. Pacharoen, T. Aoki, A. Surarerks, and P. Bhattarakosol, "Back to results conformance verification between web service choreography and implementation using learning and model checking," in *Proceedings of the 2011 IEEE International Conference on Web Services*. Washington, DC, USA: IEEE, July 2011, pp. 722 – 723.

[245] M. Palacios, J. García-Fanjul, J. Tuya, and C. de la Riva, "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, August 2010, pp. 453–458.

[246] M. Palacios, J. García-Fanjul, and J. Tuya, "Testing in service oriented architectures with dynamic binding: A mapping study," *Information and Software Technology*, vol. 53, no. 3, pp. 171–189, March 2011.

[247] M. Palomo-Duarte, A. García-Domínguez, I. Medina-Bulo, A. Álvarez Ayllón, and J. Santacruz, "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, pp. 531–534.

[248] P. Papapanagiotou and J. Fleuriot, "Formal verification of web services composition using linear logic and the pi-calculus," in *Proceedings of the 20119th IEEE European Conference on Web Services (ECOWS)*. Lugano, Switzerland: IEEE, September 2011, pp. 31 – 38.

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

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

[251] M. P. Papazoglou and J. jacques Dubray, “A survey of web service technologies,” University of Trento, Tech. Rep., Jun. 2004. [Online]. Available: http://eprints.biblio.unitn.it/archive/00000586/;http://eprints.biblio.unitn.it/archive/00000586/01/mike.pdf

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

[253] Y. Park, W. Jung, B. Lee, and C. Wu, “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, July 2009, pp. 107–114.

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

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

[256] Petri Net Markup Language (PNML). Accessed: 27.03.2012. [Online]. Available: http://www2.informatik.hu-berlin.de/top/pnml/about.html

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

[258] D. Pilone and N. Pitman, *UML 2.0 in a Nutshell (In a Nutshell (O’Reilly))*. Sebastopol, CA, USA: O’Reilly Media, Inc., 2005.

[259] G. H. L. Pinto and S. R. Vergilio, “A multi-objective genetic algorithm to test data generation,” in *ICTAI ’10: Proceedings of the 22th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1. Arras, France: IEEE, October 2010, pp. 129–134.

[260] PLASTIC Framework. Accessed: 27.03.2012. [Online]. Available: http://plastic.isti.cnr.it/wiki/tools

[261] V. Pretre, F. Bouquet, and C. Lang, “Using common criteria to assess quality of web services,” in *ICSTW ’09: Proceedings of the 2nd IEEE International Conference on Software Testing, Verification, and Validation Workshops*. Denver, CO, USA: IEEE Computer Society, 2009, pp. 295–302.

[262] Promela Manual. Accessed: 27.03.2012. [Online]. Available: http://spinroot.com/spin/Man/promela.html

[263] Z. Qi, L. Liu, F. Zhang, H. Guan, H. Wang, and Y. Chen, “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, July 2009, pp. 171–178.

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

[265] P. Ramsokul and A. Sowmya, “A sniffer based approach to ws protocols conformance checking,” in *ISPDC ’06: Proceedings of The 5th International Symposium on Parallel and Distributed Computing*. Timisoara, Romania: IEEE Computer Society, July 2006, pp. 58–65.

[266] O. Rana, A. Akram, R. A. Ali, D. Walker, G. von Laszewski, and K. Amin, “Quality-of-service based grid communities,” in *Extending Web Services Tech-*

nologies, ser. Multiagent Systems, Artificial Societies, and Simulated Organizations, L. Cavedon, Z. Maamar, D. Martin, B. Benatallah, and G. Weiss, Eds. Berlin, Heidelberg: SpringerLink, August 2005, vol. 13, pp. 161–186.

[267] Random.org integer generator. Accessed: 27.07.2012. [Online]. Available: http://www.random.org/clients/http/

[268] Remote Methods. Accessed: 27.03.2012. [Online]. Available: http://www.remotemethods.com/

[269] Resource Description Framework (RDF). Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/RDF/

[270] H. Reza and D. Van Gilst, "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, April 2010, pp. 216–221.

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

[272] K. Y. Rozier, "Linear temporal logic symbolic model checking," *Computer Science Review*, vol. 5, no. 2, pp. 163–203, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574013710000407

[273] S. Rubel. (2008, March) The future is web services, not web sites. Accessed: 27.03.2012. [Online]. Available: http://www.micropersuasion.com/2008/03/the-future-is-w.html

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

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

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

[277] R. Sagarna and X. Yao, "Handling constraints for search based software test data generation," in *ICST '08: Proceedings of the 1st IEEE International Conference on Software Testing, Verification and Validation*. Lillehammer, Norway: IEEE, April 2008, pp. 232–240.

[278] I. Saleh, G. Kulczycki, and M. B. Blake, "Formal specification and verification of data-centric service composition," in *ICWS '10: Proceedings of the 2010 IEEE International Conference on Web Services*. Miami,FL,USA: IEEE Computer Society, July 2010, pp. 131–138.

[279] S. Salva and I. Rabhi, "A preliminary study on BPEL process testability," in *ICSTW '10: Proceedings of the 3rd IEEE International Conference on Software Testing, Verification, and Validation Workshops*. Paris, France: IEEE Computer Society, April 2010, pp. 62–71.

[280] S. Salva and I. Rabhi, "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, May 2010, pp. 167–173.

[281] N. Sasikaladevi and L. Arockiam, "Correctness evaluation model for composite web service," in *TISC 2011: Proceedings of the 3rd International Conference on Trendz in Information Sciences and Computing*, 2011.

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

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

[284] Selfseo. Find ip address of a website. Accessed: 27.03.2012. [Online]. Available: http://www.selfseo.com/find_ip_address_of_a_website.php

[285] Semantic Web Services Language (SWSL). Accessed: 27.03.2012. [Online]. Available: http://www.daml.org/services/swsl/

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

[287] S. Shafin, L. Zhang, and X. Xu, “Automated testing of web services system based on OWL-S,” in *WICT 2012: Proceeding of the 2012 World Congress on Information and Communication Technologies*. Trivandrum, India: IEEE, October 2012, pp. 1103–1108.

[288] A. Sharma, T. D. Hellmann, and F. Maurer, “Testing of web services – a systematic mapping,” in *SERVICES ’12: Proceedings of the 2012 IEEE 8th World Congress on Services*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 346–352.

[289] Shipping Calculator. Accessed: 27.07.2012. [Online]. Available: http://www.unitedstateszipcodes.org/shipping-calculator/

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

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

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

[293] A. Skonnard. (2003, April) Web services and datasets. MSDN magazine. Accessed: 27.03.2012. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc188755.aspx

[294] C. Smythe, "Initial investigations into interoperability testing of web services from their specification using the unified modelling language," in *Proceedings of the International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, A. Bertolino and A. Polini, Eds., Palermo, Italy, June 2006, pp. 95–119. [Online]. Available: www.selab.isti.cnr.it/ws-mate/WS-MaTe_Proceedings.pdf

[295] H. M. Sneed and S. Huang, "WSDLTest - a tool for testing web services," in *WSE '06: Proceedings of the 8th IEEE International Symposium on Web Site Evolution*. Philadelphia, PA, USA: IEEE Computer Society, Sept. 2006, pp. 14–21.

[296] SOAP Sonar. Accessed: 27.03.2012. [Online]. Available: http://www.crosschecknet.com/products/soapsonar.php

[297] SOAP Version 1.2. Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/TR/soap12-part1/

[298] SoapUI. Accessed: 27.03.2012. [Online]. Available: http://www.soapui.org/

[299] SOATest. Accessed: 27.03.2012. [Online]. Available: http://www.parasoft.com/jsp/products/soatest.jsp?itemId=101

[300] SPIN. Accessed: 27.03.2012. [Online]. Available: http://spinroot.com/spin/whatispin.html

[301] H. Srikanth and M. Cohen, "Regression testing in software as a service: An industrial case study," in *ICSM 2011: Proceedings of the 27th IEEE International Conference on Software Maintenance*. Williamsburg, VI, USA: IEEE, September 2011, pp. 372–381.

[302] A. Stefanescu, M.-F. Wendland, and S. Wieczorek, "Using the UML testing profile for enterprise service choreographies," in *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, vol. 0. Lille, France: IEEE Computer Society, 2010, pp. 12–19.

[303] A. Stefanescu, S. Wieczorek, and A. Kirshin, "MBT4Chor: A model-based testing approach for service choreographies," in *Model Driven Architecture - Foundations and Applications*, ser. Lecture Notes in Computer Science, R. Paige, A. Hartman, and A. Rensink, Eds. Berlin / Heidelberg: Springer-Verlag, 2009, vol. 5562, pp. 313–324.

[304] C. Sun, Y. Shang, Y. Zhao, and T. Y. Chen, "Scenario-oriented testing for web service compositions using BPEL," in *QSIC 2012: Proceedings of the 12th International Conference on Quality Software*. Xi'an, China: IEEE, August 2012, pp. 171–174.

[305] C. Sun, Y. Shang, Y. Zhao, and T. Y. Chen, "Towards dynamic random testing for web services," in *COMPSAC 2012: Proceedings of the IEEE 36th Annual*

Computer Software and Applications Conference. Izmir, Turkey: IEEE, July 2012, pp. 164 – 169.

[306] C. Sun, G. Wang, B. Mu, H. Liu, ZhaoShunWang, and T. Chen, "Metamorphic testing for web services: Framework and a case study," in *ICWS 2011: Proceedings of the 2011 IEEE International Conference on Web Services*. Washington, DC, USA: IEEE, July 2011, pp. 283–290.

[307] C. Sun, Y. Zhai, Y. Shang, and Z. Zhang, "Toward effectively locating integration-level faults in BPEL programs," in *QSIC 2012: Proceedings of the 12th International Conference on Quality Software*. Xi'an, China: IEEE, August 2012, pp. 17 – 20.

[308] Swoogle. Semantic web search. Accessed: 27.03.2012. [Online]. Available: http://swoogle.umbc.edu/

[309] SWRL: A Semantic Web Rule Language. Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/Submission/SWRL/

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

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

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

[313] A. Tarhini, H. Fouchal, and N. Mansour, "A simple approach for testing web service based applications," in *Proceedings of the 5th International Workshop on Innovative Internet Community Systems (IICS 2005)*, ser. Lecture Notes in Computer Science, A. Bui, M. Bui, T. Böhme, and H. Unger, Eds., vol. 3908. Springer, 2005, pp. 134–146. [Online]. Available: http://dx.doi.org/10.1007/11749776_12

[314] S. Thummalapenta, T. Xie, N. Tillmann, J. de Halleux, and Z. Su, "Synthesizing method sequences for high-coverage testing," in *OOPSLA '11: Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications*. New York, NY, USA: ACM, 2011, pp. 189–206.

[315] S. Thummalapenta, T. Xie, N. Tillmann, P. de Halleux, and W. Schulte, "MSeqGen: Object-oriented unit-test generation via mining source code," 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 on European Software Engineering Conference and Foundations of Software Engineering Symposium*. Amsterdam, The Netherlands: ACM Press., August 2009, pp. 193–202.

[316] V. Todica, M.-F. Vaida, and M. Cremene, "Formal verification in web services composition," in *Proceedings of the 2012 IEEE International Conference on Automation Quality and Testing Robotics (AQTR)*. Cluj-Napoca, Romania: IEEE, May 2012, pp. 195 – 200.

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

[318] W. T. Tsai, Y. Chen, R. Paul, N. Liao, and H. Huang, "Cooperative and group testing in verification of dynamic composite web services," in *COMPSAC '04:*

Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts, vol. 2. Hong Kong, China: IEEE Computer Society, Sept. 2004, pp. 170–173.

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

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

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

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

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

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

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

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

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

[328] W.-T. Tsai, P. Zhong, J. Balasooriya, Y. Chen, X. Bai, and J. Elston, "An approach for service composition and testing for cloud computing and testing for cloud computing," in *ISADS 2011: Proceedings of the 10th International Symposium on Autonomous Decentralized Systems*. Tokyo & Hiroshima, Japan: IEEE, March 2011, pp. 631 – 636.

[329] UDDI Spec Technical Committee Draft. Accessed: 27.03.2012. [Online]. Available: http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm

[330] UPS Calculate Time and Cost. Accessed: 27.07.2012. [Online]. Available: https://wwwapps.ups.com/ctc/request?loc=en_US

[331] USPS Postage Price Calculator. Accessed: 27.07.2012. [Online]. Available: http://postcalc.usps.com/

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

[333] I. Vlaev, N. Chater, R. Lewis, and G. Davies, "Reason-based judgments: Using reasons to decouple perceived price-quality correlation," *Journal of Economic Psychology*, vol. 30, no. 5, pp. 721–731, October 2009.

[334] W. Wan Ab. Rahman and F. Meziane, "Challenges to describe QoS requirements for web services quality prediction to support web services interoperability in electronic commerce," in *IBIMA '08: Proceedings of 10th IBIMA conference on Innovation and Knowledge Management in Business Globalization*, vol. 4, no. 6. Kuala Lumpur, Malaysia: International Business Information Management Association (IBIMA), June 2008, pp. 50–58.

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

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

[337] X. Wang, N. Huang, and R. Wang, "Mutation test based on OWL-S requirement model," in *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*. Los Angeles, CA, USA: IEEE Computer Society, July 2009, pp. 1006–1007.

[338] Y. Wang, X. Bai, J. Li, and R. Huang, "Ontology-based test case generation for testing web services," in *ISADS '07: Proceedings of the Eighth International*

Symposium on Autonomous Decentralized Systems. Sedona, AZ, USA: IEEE Computer Society, Mar. 2007, pp. 43–50.

[339] Y. Wang, F. Ishikawa, and S. Honiden, "Business semantics centric reliability testing for web services in BPEL," in *SERVICES '10: Proceedings of the 2010 6th World Congress on Services*, ser. SERVICES '10. Los Angeles, CA, USA: IEEE Computer Society, July 2010, pp. 237–244.

[340] K. Z. Watkins, "Introducing fault-based combinatorial testing to web services," in *Proceedings of the IEEE SoutheastCon 2010*. Charlotte-Concord, NC, USA: IEEE Computer Society, March 2010, pp. 131–134.

[341] Web Ontology Language (OWL). Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/2004/OWL/

[342] Web Service Interoperability Organisation (WS-I). Basic Profile 1.2. Accessed: 27.03.2012. [Online]. Available: http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile

[343] Web Service Modelling Ontology (WSMO). Accessed: 27.03.2012. [Online]. Available: http://www.wsmo.org/

[344] Web Service Semantics (WSDL-S). Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/Submission/WSDL-S/

[345] Web Services Addressing (WS-Addressing). Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/Submission/ws-addressing/

[346] Web Services Agreement Specication (WSAgreement). Accessed: 27.03.2012. [Online]. Available: www.ogf.org/documents/GFD.107.pdf

[347] Web Services Atomic Transaction (WS-AtomicTransaction). Accessed: 27.03.2012. [Online]. Available: http://schemas.xmlsoap.org/ws/2004/10/wsat/

[348] Web Services Business Activity (WS-BusinessActivity). Accessed: 27.03.2012. [Online]. Available: http://docs.oasis-open.org/ws-tx/wsba/2006/06

[349] Web Services Description Language (WSDL 1.1). Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/TR/wsdl

[350] Web Services Glossary. Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/TR/ws-gloss/

[351] Web Services Metadata Exchange (WS-MetadataExchange). Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/TR/2009/WD-ws-metadata-exchange-20090317/

[352] J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," *Information and Software Technology*, vol. 43, no. 14, pp. 841 – 854, 2001.

[353] E. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 703–711, Jul 1991.

[354] S. Wieczorek, V. Kozyura, A. Roth, M. Leuschel, J. Bendisposto, D. Plagge, and I. Schieferdecker, "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, 2009, pp. 179–194.

[355] Wintergreen Research, Inc. (2006) Services oriented architecture (SOA) market opportunities, strategies, and forecasts, 2006 to 2012. Accessed: 27.03.2012. [Online]. Available: http://www.wintergreenresearch.com/reports/soa.html

[356] WofBPEL and BPEL2PNML. Accessed: 27.03.2012. [Online]. Available: http://www.bpm.fit.qut.edu.au/projects/babel/tools/

[357] WSDL2Java. Accessed: 27.03.2012. [Online]. Available: http://cwiki.apache.org/CXF20DOC/wsdl-to-java.html

[358] WSDL2OWL-S Project. Accessed: 27.03.2012. [Online]. Available: http://www.semwebcentral.org/projects/wsdl2owl-s/

[359] wsrbench. Accessed: 27.03.2012. [Online]. Available: http://wsrbench.dei.uc.pt/

[360] C.-S. Wu and Y.-T. Lee, "Automatic saas test cases generation based on soa in the cloud service," in *CloudCom 2012: Proceedings of the IEEE 4th International Conference on Cloud Computing Technology and Science*. Tapei, Taiwan: IEE, December 2012, pp. 349 – 354.

[361] X. Xiao, "Problem identification for structural test generation: First step towards cooperative developer testing," in *ICSE '11: Proceedings of the 33rd International Conference on Software Engineering*, May 2011.

[362] XML Path Language (XPath). Accessed: 27.03.2012. [Online]. Available: http://www.w3.org/TR/xpath/

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

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

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

[366] B. Yang, J. Wu, C. Liu, and L. Xu, "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, pp. 1–4.

[367] X. Yang, J. Huang, and Y. Gong, "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, December 2010, pp. 315–321.

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

[369] K. Ye, J. Huang, Y. Gong, and X. Yang, "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, pp. V7–472 – V7–475.

[370] G. Yeom, T. Yun, and D. Min, "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, 2006, pp. 199–204.

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

[372] S. Yoo and M. Harman, "Using hybrid algorithm for pareto effcient multi-objective test suite minimisation," *Journal of Systems Software*, vol. 83, no. 4, pp. 689–701, April.

[373] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of International Symposium on Software Testing and Analysis (ISSTA 2007).* ACM Press, July 2007, pp. 140–150.

[374] S. Yoo and M. Harman, "Regression testing minimisation, selection and prioritisation: A survey," *Software Testing, Verification, and Reliability*, 2010, *To appear*.

[375] J. Yu, P. Baumann, and X. Wang, "RPRA: A novel approach to mastering geospatial web service testing complexity," in *ICSDM 2011: Proceedings of the IEEE International Conference on Spatial Data Mining and Geographical Knowledge Services.* Fuzhou, China: IEEE, June 2011, pp. 252 – 25.

[376] Y. Yu, N. Huang, and Q. Luo, "OWL-S based interaction testing of web service-based system," in *NWESP '07: Proceedings of the 3rd International Conference on Next Generation Web Services Practices.* Seoul, South Korea: IEEE Computer Society, Oct. 2007, pp. 31–34.

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

[378] M. Yuan, Z. Huang, X. Li, and Y. Yan, "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, 2010, pp. 361–368.

[379] Y. Yuan, Z. Li, and W. Sun, "A graph-search based approach to BPEL4WS test generation," in *ICSEA '06: Proceedings of the International Conference on Soft-*

ware Engineering Advances. Tahiti, French Polynesia: IEEE Computer Society, Oct. 2006, p. 14.

[380] Q. Yue, X. Lu, Z. Shan, Z. Xu, H. Yu, and L. Zha, "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, July 2009, pp. 155–162.

[381] Q. Yue, Z. Xu, H. Yu, W. Li, and L. Zha, "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, July 2007, pp. 330–337.

[382] E. Zahoor, O. Perrin, and C. Godart, "Web services composition verication using satisability solving," in *ICWS 2012: proceedings of the 10th International Conference on Web Services*. Honolulu, HI, USA: IEEE, July 2012, pp. 242 – 249.

[383] K. Zhai, B. Jiang, W. K. Chan, and T. H. Tse, "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 2010 IEEE International Conference on Web Services*. Miami, FL, USA: IEEE Computer Society, July 2010, pp. 211–218.

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

[385] T. Zhang, Q. Yao, X. Zheng, C. Ma, and H. Wang, "An approach of end user regression testing for semantic web services," in *MASS 2011: Proceedings of the 2011 International Conference on Management and Service Science*. Wuhan, China: IEEE, August 2011, pp. 1–4.

[386] Y. Z. Zhang, W. Fu, and J. Y. Qian, “Automatic testing of web services in haskell platform,” *Journal of Computational Information Systems*, vol. 6, no. 9, pp. 2859–2867, 2010.

[387] Y. Zhang, M. Harman, and S. A. Mansouri, “The multi-objective next release problem,” in *GECCO ’07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, London, UK, July 2007, pp. 1129–1137.

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

[389] H. Zhao, J. Sun, and X. Liu, “A model checking based approach to automatic test suite generation for testing web services and BPEL,” in *APSCC 2012: Proceedings of the 2012 IEEE Asia-Pacific Services Computing Conference*. Guilin, China: IEEE, December 2012, pp. 61 – 69.

[390] H. Zhao, W. Wang, J. Sun, and Y. Wei, “Research on formal modeling and verification of bpel-based web service composition,” in *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, 2012.

[391] Y. Zheng, J. Zhou, and P. Krause, “Analysis of BPEL data dependencies,” in *EUROMICRO ’07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*. Lübeck, Germany: IEEE Computer Society, August 2007, pp. 351–358.

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

[393] L. Zhou, J. Ping, H. Xiao, Z. Wang, G. Pu, and Z. Ding, "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, November 2010, pp. 138–154.

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

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

[396] H. Zhu and Y. Zhang, "Collaborative testing of web services," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 116 – 130, March 2012.

[397] J. Zhu, K. Zhang, and G. Zhang, "Verifying web services composition based on ltl and colored petri net," in *Proceedings of the 2011 6th International Conference on Computer Science & Education (ICCSE).* Singapore: IEEE, August 2011, pp. 1127 – 1130.

[398] Z. Zhu, J. Li, Y. Zhao, and Z. Li, "SCENETester: A testing framework to support fault diagnosis for web service composition," in *CIT 2011: Proceedings of the 11th IEEE International Conference on Computer and Information Technology*. Pafos, Cyprus: IEEE, August 2011, pp. 109 – 114.