Combining TestFrame with Web Services

An Automated Testing Approach

Timothy Tjon Tsoe Jin
Combining TestFrame with Web Services

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Timothy Tjon Tsoe Jin
born in Paramaribo, Suriname

TU Delft
Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Logica
George Hintzenweg 89
Rotterdam, the Netherlands
www.logica.nl
Combining TestFrame with Web Services

Author: Timothy Tjon Tsoe Jin
Student id: 1150685
Email: tttj@ziggo.nl

Abstract
The software testing method currently used by the IT consultancy company Logica is the TestFrame method. This testing method is used for example on testing Web Services. Currently, the combination of the TestFrame method with Web Services is still in an infant stage. This thesis analyzes and models the current application of the TestFrame method on Web Services and defines a new method for Web Service testing in general. The new method, the Automatic Web Service Mapping (AWeSoMe) method defines various forms of automation of the Web Service testing process. This new method is combined with the TestFrame method and Web Services, generating a new approach to applying the TestFrame method on Web Services. Part of implementing the new method is implementing a proof-of-concept application, called the WSDL2TestFrame Application. A Case Study is performed in which the new approach compared to the original approach is evaluated. Evaluation is achieved by comparing the Test Results and the required manual intervention needed of both approaches. Based on the comparisons, the accuracy of both approaches could be measured against each other. The Case Study shows that the error-prone factor decreases and that in some areas development effort decreases linearly with the new approach compared to the old approach.

Thesis Committee:

Chair: Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor: Dr. S. Dulman, Faculty EEMCS, TU Delft
Company supervisor: P. Kathusing, Logica
Committee Member: Dr. Phil. H.-G. Gross, Faculty EEMCS, TU Delft
Contents

Contents iii
List of Figures v
List of Tables vi
Listings vi

1 Introduction 1
   1.1 Problem Statement ............................... 2
   1.2 Contributions .................................... 3
   1.3 Overview ......................................... 3

2 Web Services: an Overview 5

3 The TestFrame Method Applied on Web Services 9
   3.1 The Theoretical TestFrame Method ............... 9
   3.2 The TestFrame Method and Web Services in Practice 15

4 The AWeSoMe Method 19
   4.1 The Product and Process Model .................. 19
   4.2 Functionality of the AWeSoMe Method ............ 21

5 The WSDL2TestFrame Implementation of the AWeSoMe Method 23
   5.1 Diagrams ........................................... 23
   5.2 The Utilized Implementations of the WSDL2TestFrame Application 23
   5.3 The Technical Design of the WSDL2TestFrame Application ........ 27

6 Case Study and Evaluation 33
   6.1 The Case .......................................... 33
   6.2 The Original Approach ............................. 34
## Contents

6.3 The New Approach .................................................. 37
6.4 Evaluation ................................................................. 42
6.5 Threats to Validity ..................................................... 55

7 Related Work ............................................................ 57
  7.1 Web Service Testing by Parameter Variance ..................... 57
  7.2 Web Service Test Scheduling ....................................... 61

8 Conclusions and Future Work ........................................ 65
  8.1 Contributions and Conclusions ..................................... 65
  8.2 Future work .............................................................. 67

Bibliography ............................................................... 69

A The Technical Diagrams of the WSDL2TestFrame Implementation 73

B Background Information of the Considered Implementations 87

C WSDL2TestFrame User Guide .......................................... 91
  C.1 Overview ................................................................. 91
  C.2 Configuration .......................................................... 92
  C.3 Using WSDL2TestFrame ............................................. 93
  C.4 Editing Test Clusters ............................................... 94
List of Figures

3.1 Class Diagram of the Original Approach of the TestFrame Method Superimposed on Web Services ................................................ 17
3.2 Activity Diagram of a Test Cycle of the Original Approach of the TestFrame Method Superimposed on Web Services ....................... 18
5.1 Class Diagram of the New Approach ............................................. 24
5.2 Activity Diagram of a Test Cycle of the New Approach .................... 25
6.1 An excerpt from the Test Cluster for the Web Operation ‘getAllBeroepSector’ ..... 35
7.1 Overview of ongoing monitor of SOA systems ................................ 64
A.1 The first set of the Package Diagrams .......................................... 74
A.2 The second set of the Package Diagrams ....................................... 75
A.3 The overview Class Diagram ...................................................... 76
A.4 The high-layer Class Diagram .................................................... 77
A.5 The low-layer Class Diagram ..................................................... 78
A.6 The WSDL2TestFrame Class .................................................... 79
A.7 The SOAGateway Class .......................................................... 80
A.8 The XmlBeansHandler Class .................................................... 81
A.9 The XmlBeansClusterReader Class ........................................... 82
A.10 XmlBeansClusterWriter Class ................................................ 83
A.11 The XmlBeansDocument Class ................................................ 84
A.12 XmlBeansNode Class Part 1 ................................................... 85
A.13 XmlBeansNode Class Part 2 .................................................... 86
List of Tables

3.1 Examples of Test Conditions .............................................. 11
3.2 An Example of a Trivial Calculation Test Case ......................... 12
3.3 An Example of an Error Handling User Input Test Case ............... 12
4.1 The AWeSoMe Parameter .................................................. 20
5.1 The Mapping of the AWeSoMe method on the New Approach .......... 24
5.2 WSDL2TestFrame Metrics .................................................. 31
6.1 The Comparison of the Test Results of Both Approaches ............... 45
6.2 The Relative Calculated Accuracy Including and Excluding the Erroneous Action Word .................................................. 52

Listings

3.1 A soapUI SOAP Message Template .......................................... 16
6.1 An Excerpt from a converted Test Cluster in the Original Approach .... 36
6.2 An Excerpt from a converted Test Cluster in the Original Approach .... 39
6.3 An Excerpt of a Report of the New Approach ............................ 41
6.4 An Excerpt of the Output of the Script Parsing the Test Results .......... 44
Chapter 1

Introduction

The testing of software (systems) is always of great concern to the Information Technology (IT) sector. The success of software is greatly determined by the correct functioning of the software. Therefore, software has to be tested accordingly. The testing process in software development can, in extreme cases, consume more than half of the total software development cycle. Web Services are no exception.

Web Services are an implementation of Service Oriented Architecture (SOA), an upcoming architecture in software systems. SOA offers the innovative division of functionality of a system into services. This has various advantages compared to traditional components, such as improved loose-coupling between services and improved correspondence between software systems and actual business processes[8][22][19]. However, this also introduces some testing challenges that need to be overcome, such as late binding (of services), which add overall unpredictability and the unavailability of the source code of services, which requires some forms of black box testing[32]. Although this Thesis will only focus on Web Services, placing Web Services in the context of SOA now and then could prove to be useful.

As new implementations and architectures arise in the IT sector, testing approaches have to adapt accordingly. One of these testing approaches is the TestFrame method[35]. The TestFrame method is currently being used in the IT consultancy company Logica Netherlands. Development of the TestFrame method started in 1995 by the company CMG, which later fusioned with the company Logica. This newly formed company, now known as Logica, still maintains and applies the TestFrame method. The method is mainly used in the Dutch division of Logica, however parts of the method may be used in other divisions. The method is an open standard and has been described in [35]. Although the method contains coordination of testing activities, the method separates test management from the actual test process, so that the method can be coupled with various test management approaches. In particular, the method is suitable to be coupled with Risk and Requirement Based Testing, as described in [38]. The TestFrame method itself is primarily a functional testing method; tests or test cases are designed in such a way
1. Introduction

to determine the adherence of the software (to be tested) to its functional design. The TestFrame method is a functional testing method and does not go into details regarding the specific use of certain technologies or implementations, such as Web Services for example.

1.1 Problem Statement

Currently, the combination of the TestFrame method with Web Services is still in an infant stage. It is in the interest of Logica to determine how the integration of the TestFrame method and Web Services can be streamlined. This form of streamlining results in a decrease of time, effort and money spent on the core business processes of a software company. This has been one of the initial goals of this Thesis and has largely defined the scope of this Thesis. Before this can be determined however, it is essential to analyze and model the current application of the TestFrame method on Web Services. This can be formulated by the following research question:

1) How can the current approach of applying the TestFrame method on Web Services be defined in terms of products and processes?

The most straightforward way to streamline the testing process is automation of (parts of) the testing process. This is in line with various literature concerning Web Services. Currently, there is hardly any form of automation concerning the Web Service testing process of the TestFrame method and little effort has been spent on determining how parts of the testing process can be automated. Firstly however, it is wise to determine a generic constructive and orderly manner on how to automate the Web Service testing process. This can be formulated by the following research question:

2) What method can automate (parts of) the Web Service testing process?

This new method can then be combined with the TestFrame method and Web Services, generating a new approach to applying the TestFrame method on Web Services. Naturally, this new approach will be different from the current/original approach. To be able to compare the two approaches, it is essential to analyze and model this new approach. This can be formulated by the following research question:

3) How can the new approach of combining the new method, the TestFrame method and Web Services be defined in terms of products and processes?

To truly attest the feasibility of the new method and the new approach requires a proof-of-concept implementation. With this implementation, it can be assessed if the new approach will be able to generate test results similar to the original approach. This can be formulated by the following research question:
4) When applying the new approach, are the test results equally valid, reliable and correct as the original approach?

At this stage, it will still be unclear what the actual impact of the new approach will be. In general, increase of the quality or the decrease of time/effort required of the method are such improvements, but other improvements are possible. Similarly, setbacks of the new approach are also possible. This can be formulated by the following research question:

5) What are the improvements and what are the setbacks of the new approach compared to the original approach?

1.2 Contributions

Regarding the previous research questions, the contributions of this Thesis are as follows:

1) The result of the analysis of the current approach of applying the TestFrame Method on Web Services in terms of products and processes

2) The definition of the AWeSoMe method which defines various forms of automation of the Web Service testing process

3) The result of the analysis of the new approach of applying the TestFrame Method on Web Services in terms of products and processes

4/5) Performing a Case Study of the original and the new approach assessing the validity, reliability and correctness of the new approach compared to the original approach. This includes developing a proof-of-concept WSDL2TestFrame Implementation, formatting test data and formatting, interpreting and evaluating the test results

1.3 Overview

The remainder of this document is outlined as follows. In chapter 3 Web Services will be determined and explained. In chapter 3 the current application of the TestFrame method on Web Services will be analyzed. In chapter 4 details concerning a new Web Service testing method, contributed by this Thesis, will be disclosed. In chapter 5 details concerning the proof-of-concept WSDL2TestFrame Implementation of the AWeSoMe method will be disclosed. In chapter 6 a Case Study will be performed and its evaluation will be disclosed. The Case Study compares the original and new method against each other. In chapter 7 related proposed approaches found in the academic literature will be listed. Finally, in chapter 8 the contributions, conclusions, points of discussion and possible future work will be summarized.
Chapter 2

Web Services: an Overview

This chapter will determine and explain Web Services, whereas chapter 3 will explain how the TestFrame method is currently being applied on the testing process of Web Services.

Web Services are an implementation of Service Oriented Architecture (SOA). Before narrowing down to Web Services, the term SOA will first be explained. SOA, as the name implies, is an architecture based on Services. To clarify the meaning of SOA, it helps to assess the definitions of Services and SOA.

Within the field of computing, a service is described as a:

(….) meaningful activity that a computer program performs on request of another computer program. (…) a service is a remotely accessible, self-contained application module [22].

And SOA is described as a:

A Service Oriented Architecture(SOA) is a software architecture that is based on the key concepts of an application frontend, service, service repository, and service bus. A service consists of a contract, one or more interfaces, and an application [22].

In addition to these compact definitions, the concept of SOA is also meant to hide implementation details and technical infrastructure from the actual business processes, where Services or compositions of Services can be interchanged with real business services [22].

When comparing SOA with the different architectures available within the field of computing, SOA can be viewed as an evolution of Component Based Architecture (CBA) as Services have a striking resemblance to components. Compared to traditional components, SOA has the foremost potential of improving loose-coupling between services, including all of the corresponding benefits.

Although SOA is abstract and does not imply implementation details, this document only focuses on the implementation of SOA by a Web Service Architecture, with Web Services as atomic units of the architecture.
2. **Web Services: an Overview**

**Web Service Definition**

Surprisingly, the World Wide Web Consortium (W3C) mentions two official definitions of Web Services:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. [8]

and

A Web service is a software system identified by a URI (RFC 2396), whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols. [9]

Both definitions do not conflict with each other. The part that says that Web services exclusively use SOAP messages is incorrect however, as, for example, REST by HTTP can also be used or other future protocols that do not exist yet [44]. In fact, the message protocol is not required to be in eXtensible Markup language (XML)[13][8][9], thus disregarding the message protocol from the definition is prudent. The definition that will be adhered to in this document will be a combination of the previous definitions and is as follows:

A Web service is a software system, identified by a URI (RFC 2396) and designed to support interoperable machine-to-machine interaction over a network, whose public interfaces and bindings are defined and described in the machine-processable format WSDL. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its description using messages conveyed by HTTP.

These official definitions are not required for the work described in this document, but are mere for informational purposes.

**Web Service Deployment**

Deploying Web Services equals to enabling the: i) communication of Web Services, ii) registration of Web Services and iii) discovery of Web Services.
These mechanisms are defined in the following (XML) documents: i) Web Services Description Language (WSDL) [11] and ii) Universal Description Discovery and Integration [25].

However, the scope of this document is solely the consumption of Web Services and not the registration or discovery Web Services and thus solely WSDL documents.

Web Services Description Language (WSDL) has been developed by IBM, Microsoft and Ariba since 2000 by combining two service description languages: NASSL (Network Application Service Specification Language) from IBM and SDL (Service Description Language) from Microsoft.

The latest version of WSDL, released in 2007, is version 2.0.

WSDL is a predefined XML format, meant for the description of Web Services. A WSDL document consists of a set of definitions. These definitions are [11]:

- **Documentation** - An optional container meant for documentation.
- **Types** - a container for data type definitions using some type system (such as XSD).
- **Message** - an abstract, typed definition of the data being communicated.
- **Interface** - an abstract set of operations supported by one or more endpoints.
- **Binding** - a concrete protocol and data format specification for a particular port type.
- **Port** - a single endpoint defined as a combination of a binding and a network address.
- **Service** - a collection of related endpoints.

Within an Interface, a set of Operations are defined which are abstract descriptions of actions supported by the service.

Compared to traditional programming, Interfaces are analogue to function libraries, Operations are analogue to functions, whilst Messages are analogue to input or output parameters.

Although the term ‘Web’ has a general meaning, it is commonly regarded as a reference to a web of interlinked hypertext documents, which utilizes the message protocol HyperText Transfer Protocol(HTTP). While the term ‘Web’ is present within WSDL, and the terms WSDL Services and Web Services are used interchangeably, WSDL does not actually enforce the utilization of HTTP. However, the (common) utilization of SOAP or REST by HTTP does imply the utilization of HTTP.

**Web Service Communication**

The most common message protocols employed by Web Services are [44]: i) SOAP [10] and ii) REpresentational State Transfer(REST) by HyperText Transfer Protocol(HTTP) [14]. However, the scope of this document is solely the message protocol
SOAP.

SOAP is an open standard, developed under the supervision of the World Wide Web Consortium (W3C) and is an extension of XML. Development started in 2000 and has since then resulted in the latest SOAP version, which is version 1.2 [10]. SOAP is a protocol intended for exchanging structured information in a decentralized, distributed environment. SOAP used to be an abbreviation for Simple Object Access Protocol, however, this has been dropped since the release of version 1.2.
Chapter 3

The TestFrame Method Applied on Web Services

Chapter 2 determined and explained the subjects under test, namely Web Services. In contrast, this chapter will focus on how the relevant software testing method, the TestFrame method, is currently being applied on Web Services. In determining and analyzing the software testing method, a distinction will be made between the theoretical TestFrame method described in [35] (section 3.1) and the actual application of the TestFrame method on Web Services in practice (section 3.2).

3.1 The Theoretical TestFrame Method

More background and context information concerning the TestFrame method can be found in chapter 1. The TestFrame method will be described by differentiating between Product Model and Process Model.

3.1.1 Product Model

This section discusses the goals, concepts and artifacts/products of the product model. Goals of the the TestFrame method are: i) High quality-to-market and a ii) Short time-to-market. To achieve this, the method upholds the following concepts:

- **Flexibility**
  The lack of a stringent prescribed roadmap or phase model, in order to support every particular project or organization. Instead, the method focuses on maximizing the reuse of Test Cases. Furthermore, the method does not exclude any auxiliary means which can aid the testing process.

- **Structuring**
  The hierarchical decomposition of the method in the following artifacts: cluster maps, test clusters, test conditions, Test Cases and Action Words.
3. The TestFrame Method Applied on Web Services

- **Auxiliary Means**
  Various means which aid the management of the testing process. Examples are progress reports, test schedules, test data generation or a system for version/result control. Inclusion of the means occur in an ad-hoc manner. In a way, this can viewed as a way to cope with the (perceived) limits of the method.

3.1.2 Products/Artifacts

The essential elements of the Product Model are called products or artifacts.

**System Under Test**

The System Under Test (SUT) is the system that is to be tested in the testing process. The testing process consists of multiple test runs. A test run commonly is initiated by invoking the TestWare.

**TestWare**

The TestWare is the software implementation used in the testing process. This implementation is developed based on the test documentation. The TestWare is responsible for determining whether tests pass or fail.

**Cluster Maps**

Cluster Maps are the highest form of test documentation. Cluster Maps are notated in a textual tabular format, resembling a part of a spreadsheet table. Every Cluster Map contains global details, divided in three categories:

- **‘Key’ details** - cluster name, name of the SUT, test type(component test, component integration test, system test, system integration test and/or acceptance test), test department and stakeholder

- **‘Assignment’ details** - product risks, importance, quality attributes and references

- **‘Execution’ details** - testing techniques and test environment

- **‘Result’ details** - criteria of success or acceptance

A Cluster Map basically constitutes a contract between the test manager and the test team which was assigned to that particular Cluster Map. The test coordinator reviews the Cluster Map before assigning the Cluster Maps to the test analysts.
Test Clusters

A Cluster Map consists of Test Clusters, every Cluster Map consists of at least one Test Cluster. The division of the Cluster Map is ad-hoc, but usually, this division is based on: the differentiation between functional and non-functional requirements, the test type, the various functions of the system, the (data-)entities of the system, or the various departments or products of the organization which uses the SUT.

Test Conditions

A Test Cluster mainly exists of a set of Test Conditions. A Test Condition is an item or event of a component or system that could be verified by one or more Test Cases, e.g. a function, transaction, feature, quality attribute, or structural element [39]. In other words, the Test Conditions are descriptive conditions or terms which must hold true for the SUT. Some example are listed in 3.1.

<table>
<thead>
<tr>
<th>Test Condition</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A customer can only be registered with a non-existing customer ID</td>
<td></td>
</tr>
<tr>
<td>The flight departure date is at least 14 days after the booking date</td>
<td></td>
</tr>
<tr>
<td>The input field of the forms are laid out according to the pre-defined definitions in document X</td>
<td></td>
</tr>
</tbody>
</table>

Guidelines for constructing the Test Conditions are:

- Use complete, but compact sentences.
- Descriptions must be unambiguous and clear.
- Choose the right level of detail.
- Initially describe the general situation, and then the special cases.
- Also include invalid paths.
- Consider one-fold versus compound Test Conditions.
Test Cases

A Test Case is a set of input values, execution pre-conditions, expected results and execution post-conditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement [39]. In the TestFrame method, a Test Case is notated in a tabular format with Action Words on the vertical axis and parameters of the Action Words on the horizontal axis.

Action Words

An Action Word constitutes a single atomic test. An Action Word is a descriptive phrase of keywords and test data and expected results (input and output parameters). The Action Words are interpreted by special supporting scripts that are called by the control script for the test [39]. Action Words are either functional or technical, but functional Action Words are of more importance. For the remainder of the text, any mention of Action Words are references to functional Action Words. Examples are given in table 3.2 and table 3.3.

<table>
<thead>
<tr>
<th>Test case</th>
<th>CALC01C10T10 addition of integers gives a correct result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input calculation</td>
<td>3 + 4</td>
</tr>
<tr>
<td>Result</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3.3: An Example of an Error Handling User Input Test Case

<table>
<thead>
<tr>
<th>Test case</th>
<th>GBDT1C10T10 mandatory birth of date omitted, show notification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Name</td>
</tr>
<tr>
<td>Register customer</td>
<td>123</td>
</tr>
<tr>
<td>Notification</td>
<td>Birth of date is mandatory</td>
</tr>
</tbody>
</table>
The Theoretical TestFrame Method

vary or be empty/null. This enables an Action Word to cover a certain aspect of the system to be tested. Also, this tabular format is clear, concise and compact. This structured Test Case format has advantages over traditionally documented Test Cases. Disadvantages of traditionally documented Test Cases are:

- Test case documents contain too much context information and are cumbersome to produce and read
- There is a large overlap between Test Cases; similar Test Cases contain the same context information
- The large amount of context information and the large overlap between Test Cases amount to error-prone maintenance top the Test Cases

Of importance is that Action Words do not describe ways in how actions should be performed; Action Words are functional descriptions. The Action Words should be as generic as possible.

Test Findings

Unexpected results of atomic tests (Action Words) are documented in the form of Test Findings.

3.1.3 Process Model

This section discusses the actors/process roles and phases of the process model.

Actors

The various actors in the TestFrame method are: Test Manager, Test Coordinator, Test Analyst, Technical Tester, Executive Tester and Test Adviser.

Test Manager

The actor that directs test projects and is responsible for the budget, the planning, the organization of the test activities and the design of the Cluster Maps. This also includes composing test plans, in-between assessments and test reports and supervising the test process. The Cluster Maps are passed on to the Test Coordinators.

Test Coordinator

The actor leading test teams within a single test projects and is, likewise the Test Manager, responsible for the budget, the planning and the organization of the test activities. Cluster Maps passed on by the Test Manager, are reviewed by the Test Coordinator and possibly reorganized in a new set of Cluster Maps. These revised Cluster Maps are decomposed in various Test Clusters. These Test Clusters are passed on to the Test Analysts.
3. The TestFrame Method Applied on Web Services

Test Analyst

The actor responsible for the analysis of the Test Clusters. The analysis, in light of the system specifications, results in the design of the Test Conditions, the Test Cases (including Action Words) and the Test Script, which defines the order in which Test Cases are executed. The Test Conditions, the Test Cases and the Test Script are passed on to the Technical and Executive Tester.

Technical Tester

The actor responsible for producing the TestWare. TestWare consists of the artifacts produced during the test process required to plan, design, and execute tests, such as documentation, scripts, inputs, expected results, set-up and clear-up procedures, files, databases, environment, and any additional software or utilities used in testing, including the Test Findings [39].

Executive Tester

The actor responsible for executing the actual tests or test runs.

Test Adviser

The actor that advises and supports the other actors in applying the methods, techniques and auxiliary means in the most optimal way.

The Test Analyst and Tester should adhere to the techniques, methods and/or tools prescribed by Test Coordinators and Test Managers. The Test Analyst and Executive Tester are responsible for documenting the test results and the tools used during the test process. Test Coordinators and Test Managers are responsible for reporting the progress of the test process and the quality of the SUT.

Note that a person can assume multiple actors. For example, it is often the case that the role of Executive Tester is assumed by a person also responsible for the role as Test Analyst or Technical Tester.

Phases

The phases that we can distinguish in the process model are:

- **Test Preparation**
  The process of defining the Cluster Maps and handing them over to the Test Coordinator and Test Analysts by the Test Manager.

- **Test Analysis**
  The process of defining the Test Clusters, Test Conditions and Test Cases, based on the Cluster Maps.
• **Test Navigation**
  The process of implementing the TestWare scripts and implementation belonging to each of the Test Cases.

• **Test Execution**
  The process of executing the Test Cases, in which the actual results or values are compared with the expected results or values. The Test Results are recorded in a Test Report and Test Findings are added to the Test Findings repository. Finally, an evaluation will take place of the test process as a whole.

### 3.2 The TestFrame Method and Web Services in Practice

After some analysis, it appeared that the application of the TestFrame method on Web Services in practice differed from the theoretical TestFrame method described in [35]. This section contains these discrepancies.

#### 3.2.1 The TestFrame Engine

In general, a couple of tools have been developed to support the general TestFrame method, in particular, the TestFrame Toolbar and the TestFrame Engine. The TestFrame Toolbar is a Microsoft Excel toolbar capable of generating a simple generic Test Cluster template in Microsoft Excel. The TestFrame Engine requires some additional explanation.

The TestFrame Engine relieves some of the effort needed for developing the TestWare Implementation, it basically 1) handles the (pre-selected) input Test Cluster, 2) handles the output TestFrame Report and 3) compares test values (actual value versus expected value). For the input Test Cluster, the TestFrame Engine is responsible for reading the Test Cluster Action Word by Action Word similar to a combination of a FileReader and StreamTokenizer Java Class with support for Test Cluster-specific delimiters. To retrieve these contents, the TestFrame Engine provides methods from an interface Class. Examples of the these methods are: getActionWord(), getActionWordFunction() and getParameter(int). The TestFrame Engine has support for Oracle Java, C(++) and HP’s Quick Test Professional among others. For the output TestFrame Report, the TestFrame Engine is responsible for generating a Rich Text Format (RTF) document containing the test results of the Test Cluster categorized by Action Words, Test Cases and Test Conditions. For comparing the test values, the result is stored for later use when generating the TestFrame Report. The TestFrame Report is generated by calling an equally named method from the interface Class.

#### 3.2.2 Differences with the Theoretical TestFrame Method

After analyzing actual Cluster Maps from the Case Study, it appeared that Test Conditions and Test Cases are very minimal and are only used descriptively. In other words, the Test Conditions and Test Cases did not contain any (execution)
3. The TestFrame Method Applied on Web Services

pre- and post-conditions or terms. Off course, this could be due to the intrinsic nature of Web Services, which are like-wise not specified in terms of pre- or post-conditions.

soapUI and SOAP Templates

The developed TestWare currently uses XML templates generated by soapUI[1]. The Web Services to be tested use SOAP as its message protocol. These SOAP messages have to be defined for the various Test Cases. To generate these messages, the application soapUI is currently being used. This approach has the same inherent advantages as soapUI, as listed in chapter 7.

The Input and Output SOAP messages of the Web Operations are determined by using the (stand-alone) application SoapUI[1]. This application determines the Input and Output SOAP messages of a Web Service by reading the WSDL document and stores templates of these SOAP messages as plain text files. The SOAP message template of the Web Operation ‘findCursus’ of the Web Service from the Case Study in chapter 6 is displayed in listing 3.1.

Listing 3.1: A soapUI SOAP Message Template

```xml
<?xml version="1.0"?>
<soapenv:Header/>
<soapenv:Body>
<ser:findCursus>
<Zoekterms>
<Zoekterm>[VAR_1]</Zoekterm>
<Zoekterm>[VAR_2]</Zoekterm>
</Zoekterms>
<CdLesplaats>[VAR_3]</CdLesplaats>
<LetterProvincie>[VAR_4]</LetterProvincie>
<CdOpleidingsinstelling>[VAR_5]</CdOpleidingsinstelling>
<CdOpleidingVorm>[VAR_7]</CdOpleidingVorm>
<CdIetsMet>[VAR_8]</CdIetsMet>
<CdSector>[VAR_9]</CdSector>
<CdBeroepsnaams>
<CdBeroepsnaam>[VAR_10]</CdBeroepsnaam>
</CdBeroepsnaams>
<MaximumAant>[VAR_11]</MaximumAant>
</ser:findCursus>
</soapenv:Body>
</soapenv:Envelope>
```

Within such a template, the (instance-specific) XML values are replaced by the (textual) ‘VAR_x’ where x is an integer index starting from 1 and is within every template. Sending a certain SOAP message occurs by reading the corresponding template, storing this into a String, inserting desired values by searching and replacing the various ‘VAR_x’ in this String, converting the string to the UTF-8 locale and passing this String to the HTTPService responsible for sending the actual SOAP message.
Figure 3.1: Class Diagram of the Original Approach of the TestFrame Method Superimposed on Web Services

Web Service Under Test
As the SUT is a Web Service, the term Web Service Under Test (WSUT) will be maintained from now on.

3.2.3 Diagrams
In summary, Figure 3.1 and 3.2 represent the relevant Class Diagram and Activity Diagram of the Product Model and Process Model of the TestFrame method superimposed on Web Services. Within the Activity Diagram, the colour yellow denotes manual Activities, dull green denotes (IT) tool-assisted Activities and bright green denotes automated Activities.
Figure 3.2: Activity Diagram of a Test Cycle of the Original Approach of the TestFrame Method Superimposed on Web Services
Chapter 4

The AWeSoMe Method

This chapter contains details concerning a new Web Services testing method, contributed by this Thesis. The method is called the AWeSoMe which stands for Automatic Web Service Mapping. The method aims to automate a number of processes or facets present in the method described in chapter 3. As a result, the method only targets (to be tested) Web Services.

4.1 The Product and Process Model

This section contains details concerning the Product and Process model of the AWeSoMe method.

4.1.1 Product Model

The AWeSoMe method consists of the AWeSoMe Test and AWeSoMe Parameter products/artifacts.

AWeSoMe Test (Template)

The AWeSoMe Test consist of the identifier/name of the to be tested Web Operation and a collection of AWeSoMe Parameters. AWeSoMe Test Templates are AWeSoMe Tests with default generated Test Values, in contrast to manually populated Test Values.

AWeSoMe Parameter

The AWeSoMe Parameter consists of three elements, a Parameter Type, a Parameter Path and a Parameter Value. The Parameter Type represents either Input or Output corresponding to the input or output message of a Web Operation. The Parameter Path represents an unambiguously hierarchical path to the desired XML/SOAP element in XML/SOAP message. Parameters in a sequential data-structure are supported, as an index can be used to uniquely define elements in the data-structure.
Parameter value represents the input value of an XML/SOAP element of an Input Parameter or the expected value of an XML/SOAP element of an Output Parameter.

<table>
<thead>
<tr>
<th>AWeSoMe Parameter</th>
<th>Parameter Type</th>
<th>Parameter Path</th>
<th>Parameter Value</th>
</tr>
</thead>
</table>

From every Input Parameter of an AWeSoMe Test, corresponding SOAP/Elements are created, populated and placed in the input message of the Web Operation. After invoking the Web Operation, every expected Parameter Value from the Output Parameters are compared to the actual output of the Web Operation. From the results of the comparisons a conclusion can be made regarding the (AWeSoMe) Test.

**AWeSoMe Test Result**

The AWeSoMe Test Result is the verdict (Passed or Failed) of an AWeSoMe Output Parameter.

**AWeSoMe TestWare**

The AWeSoMe TestWare is the Implementation capable of performing the AWeSoMe processes based on a Web Service, a WSDL document and a collection of AWeSoMe Tests.

### 4.1.2 Process Model

The processes that the AWeSoMe method entails are as follows:

**Automatic generation of AWeSoMe Test Templates including AWeSoMe Tests and AWeSoMe Parameters**

A collection of AWeSoMe Test Templates are automatically generated based on the WSDL document of a Web Service by the AWeSoMe TestWare. At least one AWeSoMe Test Template should be generated for each Web Operation (defined in the WSDL document). The AWeSoMe Test format is in such a way that the AWeSoMe TestWare can perform a Test Run based on these AWeSoMe Tests in an automated fashion without any additional TestWare besides the AWeSoMe TestWare. The Input/Output Parameters of the AWeSoMe Tests can be populated with values based on the XML Type. These values can be constants, random or more refined generated values. The AWeSoMe Test Templates or copies thereof can of course be manually customized with other desired values.
Manual populating the generated Input/Output Parameters of the AWeSoMe Tests of Web Services

The user inspects the generated Test Clusters of Use Case 1 and populates the generated data values with custom values if necessary. Action Words can be duplicated in order to test additional data values.

Automatic handling of the Input Parameters of AWeSoMe Tests of Web Services

The process of handling AWeSoMe Tests in such a way that input messages of Web Operations are always constructed in an unambiguously and automated manner. This process includes parsing the Input Parameters and the automatic generation and population of the input messages of Web Operations based on Input Parameters.

Automatic Test Execution of Web Services based on AWeSoMe Tests of Web Services

The process of performing a set of tests related to (a set of) AWeSoMe Tests solely based on the WSDL document, the set AWeSoMe Tests and AWeSoMe TestWare. No additional TestWare is required besides the AWeSoMe TestWare itself. This process includes automatic invoking the referred Web Operation in each AWeSoMe Test with the generated and populated input message.

Automatic handling of the Output Parameters of AWeSoMe Tests of Web Services

The process of handling AWeSoMe Tests in such a way that the output message of a Web Operation can be validated against the Output Parameters in an unambiguously and automated manner. This process includes parsing and interpreting the Output Parameters, comparing them with the elements of the output messages of Web Operations and generating an AWeSoMe Test Result.

Automatic generation of the AWeSoMe Test Results of AWeSoMe Tests of Web Services

The process of generating and listing the AWeSoMe Test Results of AWeSoMe Tests in an automated manner without any additional implementation required besides the AWeSoMe TestWare.

4.2 Functionality of the AWeSoMe Method

the AWeSoMe Method contains 3 distinct Use Cases 1) Test Generation, 2) Test Population and 3) Test Execution. All the processes of the AWeSoMe method are embedded in these Use Cases. Any system implementing the AWeSoMe Method must implement 1) Test Generation and 2) Test Execution.
Use Case 1: Test Generation

The Use Case is as follows:

1. The user selects the desired WSDL document of the to be tested Web Service, preferably selectable from a list of WSDL documents.
2. The user selects a desired container containing a set of AWesoMe Tests, preferably selectable from a list of containers or opts for an automatically generated container.
3. The user signals the system to start the Test Generation process.
4. The system generates the any necessary Web Service-specific stub code package and databinding stub code package, based on the Web Operations and parameters in the WSDL Document.
5. The system outputs a Test Cluster to the specified output file with an Action Word for every deduced Web Operation combined with its parameters. Generated (expected) test data values are generated depending on the specific data type (string, integer, date, etc).

Use Case 2: Test Population

The Use Case is as follows:

1. The user inspects the generated Test Clusters of Use Case 1 and populates the generated data values with custom values if necessary. Action Words can be duplicated in order to test additional data values.

Use Case 3: Test Execution

The Use Case is as follows:

1. The user selects the desired (pre-generated) stub code package, preferably selectable from a list of stub code packages.
2. The user selects the desired container containing a set of AWesoMe Tests, preferably selectable from a list of containers.
3. The user signals the system to start the Test Execution process.
4. The system reads the Actions Words from the input Test Cluster and tests each Action Word. Each Action Word is tested by invoking the corresponding Web Operations with the corresponding parameters, which is accomplished by invoking the corresponding functionality from the stub code package with the corresponding data-structures.
5. The system evaluates the Output Parameters of every AWesoMe Test and generates a container of corresponding AWesoMe Tests.
Chapter 5

The WSDL2TestFrame Implementation of the AWeSoMe Method

This Thesis provides not only the AWeSoMe method, but also an implementation of the AWeSoMe method itself. The implementation serves to attest the viability of the AWeSoMe method a la proof of concept. This implementation has been developed under the context of the TestFrame method combined with Web Services and is called the WSDL2TestFrame Application. The application enables the user to perform and generate all the processes and products/artifacts of the AWeSoMe method. The combination of the TestFrame method, the AWeSoMe method, the WSDL2TestFrame application and Web Services has resulted in a new approach of Web Service testing.

5.1 Diagrams

The new approach naturally resulted in a different Process and Product Model than the ones defined in 3. Specifically, the processes ‘Implement TestWare’ and ‘Generate SOAP Templates’ and the products/artifacts TestWare, soapUI and SOAP Templates have been rendered obsolete. Table 5.1 contains the mapping of the AWeSoMe onto the new approach. Figure 5.1 and 5.2 represent the new Class and Activity Diagrams.

5.2 The Utilized Implementations of the WSDL2TestFrame Application

Before the WSDL2TestFrame Application can be implemented, the proper existing implementations need to be determined upon which the WSDL2TestFrame Ap-
5. The WSDL2TestFrame Implementation of the AWeSoMe Method

Table 5.1: The Mapping of the AWeSoMe method on the New Approach

<table>
<thead>
<tr>
<th>AWeSoMe</th>
<th>WSDL2TestFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container of AWeSoMe Tests</td>
<td>(WSDL2TestFrame) Test Cluster</td>
</tr>
<tr>
<td>AWeSoMe Test</td>
<td>Action Word</td>
</tr>
<tr>
<td>AWeSoMe (Input/Output) Parameter</td>
<td>(Input/Output) Parameter</td>
</tr>
<tr>
<td>AWeSoMe Test Result</td>
<td>Test Result</td>
</tr>
<tr>
<td>Container of AWeSoMe Test Result</td>
<td>TestFrame Report</td>
</tr>
<tr>
<td>AWeSoMe TestWare</td>
<td>WSDL2TestFrame Application</td>
</tr>
</tbody>
</table>

Figure 5.1: Class Diagram of the New Approach

plication will be build. To determine this, a survey of already existing software implementations which contain overlapping aspects with the WSDL2TestFrame Application needs to be made. The most suitable implementations can then serve as a basis when developing the WSDL2TestFrame Application. A multitude of implementations exist which contain overlapping aspects of the WSDL2TestFrame Application. Because it is in the interest of Logica to retain the right on how to
publish the WSDL2TestFrame Application, this restricts us to software under the Apache License, Mozilla License, Lesser General Public license or any other similar license. With this in mind, the following implementations have been considered:

- The Apache Axis and Axis2 framework
- The java.net JAX-WS Reference Implementation
- The Apache CXF framework
- WS-TAXI
5. The WSDL2TestFrame Implementation of the AWeSoMe Method

- SoapUI

Other less prominent implementations that were also considered, but did not have a compatible license, did not have an explicitly (public) license, are no longer developed and lacking in features, did not support WSDL 2.0 or are a combination of aforementioned, are: ActiveSOAP[6] and [36].

Some of the considered implementations require a data binding framework. The relevant set of data binding frameworks is as follows[17][21]:

- Axis2 Data Binding (ADB)
- XMLBeans
- JiBX
- JAXB (RI)

Appendix B contains the relevant background information of the listed implementations and data binding frameworks.

Although WS-TAXI and SoapUI accomplish some goals common with the WSDL2TestFrame Application, they have some unfavorable drawbacks. The source code of WS-TAXI is viewable to the public, but any documentation, external or in-code, is lacking. On top of this, WS-TAXI could not be run directly out of the box. Some configuration or imports were probably needed. With no documentation and with no knowledge of which license the code is under, WS-TAXI would not appear to be very helpful.

SoapUI does work directly out of the box, however in its current form, a stand-alone application, it can hardly be integrated with any other application. Regardless of the license, the source code is not publicly available, which denies the ability for the WSDL2TestFrame Application to interface with (parts of) SoapUI.

Based on the aforementioned advantages of Axis2 over Axis, it is clear that Axis2 is preferred over Axis.

As Apache CXF does not support WSDL version 2 and Axis2 and JAX-WS RI do, choosing CXF would limit the applicability of the WSDL2TestFrame Application.

When comparing both Axis2 and JAX-WS, there is no clear winner as they are equally powerful. A notable feat of JAX-WS is that it also provides support for WS Annotations, however WS Annotations fall out of the scope of the WSDL2TestFrame Application. A notable feat of Axis/Axis2 is that it has been developed since 2004 and already established a large user base and a long bug report/fix history. This small advantage contributed to the choice of Axis2.

Axis2, likewise to some of the other implementations, requires a data binding framework. Although the ADB framework is easier to use, it does not generate Classes which unambiguously reflect the whole structure of the XML Schema of input and output messages. Details such as the level of XML nodes are not clear. Also, more complex XML Schemas are not supported by the ADB framework, so ADB is not a good choice. The JiBX framework is a more complete data binding framework and supports more XML Schemas than ADB, but the most complete data binding
frameworks are XMLBeans and JAXB. When comparing both the XMLBeans and JAXB frameworks, again there is no clear winner as they are equally powerful. Notable feats of XMLBeans is that development of XMLBeans started earlier and was the most complete data binding framework in the past and is still Java 1.4 compliant, whereas JAXB is only Java 1.5+ compliant. Notable feats of JAXB and the reference project is that they are both under the supervision of Oracle itself and is bundled with Java since 1.6. Both feats do not justify a preference of one above the other. Regardless, one data binding framework must be chosen and as such, the XMLBeans framework has been chosen for the WSDL2TestFrame Application.

5.3 The Technical Design of the WSDL2TestFrame Application

This section contains details about the technical design of the WSDL2TestFrame Application. This entails describing the various layers of the WSDL2TestFrame Application and displaying the Class Diagrams, the Sequence Diagrams and the Deployment Diagram.

5.3.1 Layers

The design of the WSDL2TestFrame Application is divided in four main layers, one cross-layer utility package and one general miscellaneous package. The four main layers and their corresponding Java Classes are as follows:

- **the GUI**
  
  - WSDL2TestFrame and FormattedConsoleStream Classes

- **the SOA Controller**
  
  - SOAGateway Class

- **the WSDL2TestFrame Cluster Controller**
  
  - XmlBeansClusterReader, XmlBeansClusterWriter, XmlBeansHandler, ClusterParser and EngineWrapper Classes

- **the WSDL2TestFrame Cluster Model**
  
  - XmlBeansDocument, XmlBeansNode, ClusterFormatException, ClusterFormat, ParameterEntry and ParameterEntryList Classes and the ClusterEntryType and ParameterType Enumerations

The utility and general packages are as follows:

- **Utility Package**
  
  - CompilerWrapper, EntryGenerator, Launcher and Util Classes

- **General Package**
  
  - Config, InvalidModelException and TestFrameException Classes and the InvocationType Enumeration
5. The WSDL2TestFrame Implementation of the AWeSoMe Method

GUI

The Graphical User Interface which concisely conveys to the user how to perform the Generation Phase and the Test Execution Phase and also simplify the process of performing said phases. Selecting the desired WSDL document, Stub Class and Test Cluster file from a list contributes to this, as well as the color-formatted Console, which gives status, notification, warning and error messages in chronological order, which reflect the process in the back end of the WSDL2TestFrame Application, which also help in determining errors made by the user in the configuration file or Cluster files.

SOA Controller

The SOA Controller layer is responsible for controlling the classes in the lower levels responsible for handling all the SOA related functionality, which basically is the core functionality of the WSDL2TestFrame Application. In the current state, the SOA related functionality is implemented as Web Services in combination with the Apache Axis2/XMLBeans framework. If in the future however, other frameworks than XMLBeans/Axis2 will be added, this layer should also control the classes in the lower layers responsible for handling these frameworks. Functionality present in this layer are: configuring the XMLBeans/Axis2 settings, setting up logging constructs, delegating generation of Axis2/XMLBeans Classes, compilation of the generated Axis2/XMLBeans Classes, delegating WSDL processing and delegating Test Cluster reading/writing.

WSDL2TestFrame Cluster Controller

The WSDL2TestFrame Cluster Controller layer is responsible for reading/writing/interpreting/formatting/handling Test Clusters and, when doing so, constructing the skeletons of XMLBeans Classes and the XMLBeans Classes themselves as needed. Control starts with XmlBeansClusterReader and XmlBeansClusterWriter, which are responsible for respectively reading and writing Test Clusters. Reading Test Clusters are less complicated compared to writing them, as the Parameters in a Test Cluster dictate the structure of the XML nodes of every SOAP message and the XmlBeansClusterReader only has to instantiate the necessary XMLBeans Classes from the WSDL2TestFrame Cluster Model. If the structure is wrong, the XmlBeansClusterReader aborts the process, reports a error message and continues with the next structure found in the next Parameter section. When writing Test Clusters the whole structure of all the SOAP messages must first be examined, which is equal to recursively traversing all the nodes and sub-nodes of the XMLBeans/XML hierarchy of all of the SOAP messages. This layer also communicates Parameters to and from the TestFrame Engine via EngineWrapper.
The Technical Design of the WSDL2TestFrame Application

WSDL2TestFrame Cluster Model

The WSDL2TestFrame Cluster Model layer is the collection of Classes and Enumerations that form the data objects of WSDL2TestFrame. Every Test Clusters being interpreted by WSDL2TestFrame has a corresponding data cluster model, which allows WSDL2TestFrame to construct and populate various XMLBeans objects necessary for the Web Service under test.

Utility Package

Various Classes containing utility functionality. Examples are: Compiling generated XMLBeans/Axis2 Classes in run-time(CompilerWrapper), launching a Thread which accomplishes this(Launcher), generating an Entry in a HashMap in a standard way(EntryGenerator) and miscellaneous utility functionality(Utit).

Global Package

The remainder of the Classes or Enumerations which have no clear place in the package hierarchy. Config contains global configurations of various types; almost every Class has access to it, the various Exceptions are used throughout the whole global hierarchy and InvocationType also has no clear place in the global hierarchy.

Deprecated Classes

The deprecated Classes ExitTrappingSecurityManager, ExitTrappedException and EngineLoader, though still present in the software package, are no longer used.

5.3.2 Package and Class Diagrams

The Package and Class Diagrams are included in A. Figure A.1 and A.2 represent the package diagrams. Note that the hierarchy in the package diagrams does not necessarily reflect the hierarchy of the layers.

The numerous Class Diagrams are differentiated based on their level of detail. Note that the Exception-derived Classes, the various Enumerations and some other minor Classes have been left out in order to keep the diagrams more manageable. Figure A.3, A.4 and A.5 represent Class Diagrams on super-Class level; details of the inner workings of the classes are left out. Every Class Diagram contains a focused Class. Red associations flow to the focused Class, green associations flow from the focused Class and blue associations flow from and flow to the focused Class. Dashed associations reflect one or more references to members(methods, fields, etc.) of that particular Class and solid associations reflect one or more references to an instance of that particular Class.

Figure A.4 and A.5 represent Class Diagrams with each a subset of the Classes, but with greater detail than the previous Class Diagrams. Figure A.4 represents a high-layer view(the GUI and SOA Controller layer) Class Diagram focused on SOAGateway. Figure A.5 represents a low-layer view(the WSDL2TestFrame Cluster
5. **The WSDL2TestFrame Implementation of the AWeSoMe Method**

Model layer) Class Diagram focused on `XmlBeansNode`. Unfortunately, a middle-layer view (the WSDL2TestFrame Cluster Controller) has not been included as this would result in a cluttered and messy Class Diagram, which would not add any value to the whole.

Figure A.6, A.7, A.8, A.9, A.10, A.11, A.12 and A.13 represent Class Diagrams on sub-Class level; associations between Classes are left out.

### 5.3.3 Metrics

WSDL2TestFrame has been analyzed by the Eclipse Metrics Plugin[34] which resulted in various metrics. Two of these metrics are listed in table 5.2. The metrics are the Lines of Code (LOC) and the average LOC per method. The metrics are categorized in Java packages or Java Classes. The metrics for the total implementation is also listed. The Metrics generated by the Eclipse Metrics Plugin do not include empty lines, comments and Javadocs. The total lines of source code, including empty lines and comments are estimated at 5500 lines.

### 5.3.4 Deployment Diagram

As the WSDL2TestFrame Application relies on the underlying Axis2 framework to handle any external communication, the System Design of the WSDL2TestFrame Application itself is stand-alone and the trivial deployment diagram can be omitted.

### 5.3.5 Interfacing with the Axis2 Client Stub

Before the application can analyze, invoke or test a Web Service, the Web Service-specific Classes must be generated. These are generated automatically by the application during run-time when required. The Web Service-specific Classes are generated by the application by calling a method of the Axis2 framework. This generation process is generally only needed once, however if the Web Service definition has changed, (reflected by a change in its WSDL document,) the Web-Service-specific Classes must be regenerated. Part of these Classes is the (client) stub Class which serves as the interface class of that particular Web Service. Because each Web Service is different, every generated stub Class is also different and thus unknown beforehand. However, the structure of the generated code is being generated by certain heuristics. Some of these heuristics can be transcribed from the documentation by Apache, while other heuristics have to be induced from the actual generated code. This process however consumes quite some time. Simple examples of these heuristics in the Axis2 framework are: a) Web Operations map one-on-one to methods in the stub Class, and b) Web Operations can be invoked synchronously and asynchronously by two different methods; the asynchronous method can be recognized by the method name which is prefixed with the character sequence ‘start’(i.e. methods ‘RegisterCustomer’ and ‘startRegisterCustomer’).
### Table 5.2: WSDL2TestFrame Metrics

<table>
<thead>
<tr>
<th>Package/Class</th>
<th>LOC</th>
<th>Average LOC/method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total</strong></td>
<td>4740</td>
<td>3945</td>
</tr>
<tr>
<td><strong>wsdl2testframe</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FormattedConsoleStream</td>
<td>985</td>
<td>731</td>
</tr>
<tr>
<td>SOAGateway</td>
<td>386</td>
<td>269</td>
</tr>
<tr>
<td>WSDL2TestFrame</td>
<td>519</td>
<td>415</td>
</tr>
<tr>
<td><strong>wsdl2testframe.cluster</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClusterParser</td>
<td>102</td>
<td>70</td>
</tr>
<tr>
<td>XmlBeansClusterReader</td>
<td>273</td>
<td>208</td>
</tr>
<tr>
<td>XmlBeansClusterWriter</td>
<td>390</td>
<td>270</td>
</tr>
<tr>
<td>XmlBeansHandler</td>
<td>899</td>
<td>729</td>
</tr>
<tr>
<td><strong>wsdl2testframe.cluster.engine</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EngineLoader</td>
<td>195</td>
<td>128</td>
</tr>
<tr>
<td>EngineWrapper</td>
<td>157</td>
<td>104</td>
</tr>
<tr>
<td><strong>wsdl2testframe.cluster.model</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClusterEntryType</td>
<td>1539</td>
<td>1181</td>
</tr>
<tr>
<td>ClusterFormat</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>ClusterFormatException</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>ParameterEntry</td>
<td>130</td>
<td>73</td>
</tr>
<tr>
<td>ParameterEntryList</td>
<td>107</td>
<td>45</td>
</tr>
<tr>
<td>ParameterType</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>XmlBeansDocument</td>
<td>244</td>
<td>182</td>
</tr>
<tr>
<td>XmlBeansNode</td>
<td>1013</td>
<td>878</td>
</tr>
<tr>
<td><strong>wsdl2testframe.model</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Config</td>
<td>177</td>
<td>91</td>
</tr>
<tr>
<td>InvalidModelException</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>InvocationType</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>TestFrameException</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>util</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CompilerWrapper</td>
<td>19</td>
<td>8</td>
</tr>
<tr>
<td>EntryGenerator</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>ExitTrappingSecurityManager</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>Launcher</td>
<td>74</td>
<td>51</td>
</tr>
<tr>
<td>Util</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td><strong>util.model</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExitTrappedException</td>
<td>16</td>
<td>4</td>
</tr>
</tbody>
</table>
The parameters of these methods are XMLBeans subclasses specific for that Web Service. The XMLBeans heuristics are more low-level and more complex. Examples of these complex heuristics in the XMLBeans framework are: a) the getter method of a child sequence (as defined in XML by for example ‘xs:sequence’) of XmlObjects are returned as an anonymous subclassed AbstractList which forwards method invocations to the parent XmlObject, and b) in contrast to the getter method, the setter method of a list of XML elements has to be set via an Array of subclassed XmlObjects objects of which its elements are copied into parent XmlObject instead of linked.

In the Test Generation Use Case, the application analyzes the Web Service solely by analyzing the Client Stub via Java’s reflection mechanisms; the Web Service is not invoked at all. By said heuristics, the various Web Service operations with its specific parameters are determined and a basic Test Cluster will be generated which will reflect this process.

In the Test Execution Use Case, the application does invoke the Web Service by invoking its Web Operations with the specific parameters, as read from the Action Words.
Case Study and Evaluation

This chapter will discuss the test setup, test results and evaluation of the WSDL2-TestFrame Case Study.

6.1 The Case

The case study revolves around testing of the Web Service called BOCPort, which is the back end of the website http://www.werk.nl. The website and back end has been developed by Logica commissioned by the Uitvoeringsinstituut Werknemersverzekeringen (UWV) [41] which is a Dutch government institution responsible for administering the various Dutch employment insurances and facilitating the transition between unemployment and (re-)employment. The Web Service contains 20 Web Operations. The list of Web Operations is as follows:

1. findBeroep
2. findOpleidingsnaam
3. getAllBeroepSector
4. getAllGedragsCompetentie
5. getAllOpleidingNiveauBemiddeling
6. getAllOpleidingSector
7. getAllSbcBeroepGroep
8. getAllSoiOpleidingRubriek
9. getBeroepDetails
10. getBeroepRelaties
11. getBeroepSectorenForBeroep
12. getBeroepVerwantschappen
13. getBeroepsoorten
14. getOpleidingDetails
15. getOpleidingRelaties
16. getOpleidingsoorten
6. Case Study and Evaluation

17. `getReferentiebrnForSbcBeroepGroep`
18. `getRefOpForSoiOplRubriek`
19. `getSbcBeroepGroepForBeroep`
20. `getSoiOpleidingRubriekForOpleiding`

The reason that some of the Web Operations are listed with italic or strikethrough text is that the Test Results of these Web Operations could not be fully included in the evaluation. This is all explained in section 6.4.

6.2 The Original Approach

In general, when describing any approach of applying the TestFrame method, the generic information present in chapter 3 is leading. However, the more technical details of any approach differ with each case. This section will cover the technical details of the current approach not covered by the generic methodology in chapter 3. This approach will be called the original approach.

Test Clusters

As can be read in chapter 3, the act of testing a particular Web Service is documented in a Cluster Map. In this case, the Cluster Map is in Microsoft Excel format where each tab is a separate Test Cluster. This is the most common format. An excerpt from the Test Cluster for the Web Operation ‘getAllBeroepSector’ is displayed in figure 6.1.

Test Clusters can be derived from a generic (context-unaware) template generated by the (Microsoft Excel) TestFrame Toolbar. Each Web Operation of the to be tested Web Service has a corresponding Test Cluster. Each Test Cluster contains a collection of Test Conditions, which contain Test Cases which contain Action Words. Each Action Word corresponds to a single atomic Web operation invocation.

Test Cluster Scoping

Test Setup

The characteristics of the environment in which the tests took place are as follows:

- Microsoft Windows XP SP2
- Sun Java SDK and JRE version 5.0 update 11
- TestFrame Engine version 6.0 BUILD 58

Test Implementation and Execution

Every Test Cluster corresponds to a single Test Execution. A Test Execution is merely running the TestWare with as input the pre-selected Test Cluster. Before the TestWare can read the input however, the Test Clusters have to be saved in the
Figure 6.1: An excerpt from the Test Cluster for the Web Operation ‘getAllBeroepSector’

<table>
<thead>
<tr>
<th>testcondition</th>
<th>boca01C10</th>
<th>Get all active and inactive items of 'Beroep'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Comment and/or reference to specification</td>
</tr>
<tr>
<td>testcase</td>
<td>boca01C10T1</td>
<td>All active and inactive items of 'BeroepSectoren'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Comment and/or reference to specification</td>
</tr>
<tr>
<td>getAllBeroepSector</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Aantal</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;Cont</td>
</tr>
<tr>
<td></td>
<td>Output element</td>
<td>BeroepSector1</td>
</tr>
<tr>
<td></td>
<td>o: CdSector</td>
<td>o: OmsSector</td>
</tr>
<tr>
<td></td>
<td>000031142</td>
<td>Automatisering en ICT</td>
</tr>
<tr>
<td></td>
<td>o: IndSectorActief</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>&amp;Cont</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output element</td>
<td>BeroepSector1.1</td>
</tr>
<tr>
<td></td>
<td>Aantal</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;Cont</td>
</tr>
<tr>
<td></td>
<td>&amp;Cont</td>
<td></td>
</tr>
<tr>
<td></td>
<td>o: CdSector</td>
<td>o: OmsSector</td>
</tr>
<tr>
<td></td>
<td>000031143</td>
<td>Ontwerp en ontwikkeling</td>
</tr>
<tr>
<td></td>
<td>o: IndSectorActief</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>&amp;Cont</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output element</td>
<td>BeroepSector1.2</td>
</tr>
<tr>
<td></td>
<td>Aantal</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&amp;Cont</td>
</tr>
<tr>
<td></td>
<td>Output element</td>
<td>BeroepSector2</td>
</tr>
<tr>
<td></td>
<td>o: CdSector</td>
<td>o: OmsSector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o: IndSectorActief</td>
</tr>
</tbody>
</table>
simpler tab-delimited textual format such that each tab-delimited file corresponds to a single Test Cluster. The result of converting the excerpt displayed in figure 6.1 is listed in 6.1.

Listing 6.1: An Excerpt from a converted Test Cluster in the Original Approach

```
1 testcondition boca01C10
   Get all active and inactive items of 'BeroepSector' with operation 'getAllBeroepSector' < positive
   Not defined Not defined Must Test
2 testcase boca01C10T10
   All active and inactive items of 'BeroepSector' have been found. _ _ _
   Not defined Not defined Must Test
3 Comment and/or reference to specification
4 Aantal
5 getAllBeroepSector 12 &Cont
6
7 Output element
8 BeroepSector1
9
10 o: CdSector o: OmsSector o: IndSectorActief
11 &Cont 0000031142 Automatisering en ICT 1 &Cont
12
13 Output element
14 BeroepSector1.1
15
16 Aantal
17 &Cont 3 &Cont
18
19 o: CdSector o: OmsSector o: IndSectorActief
20 &Cont 0000031143 Ontwerp en ontwikkeling 1 &Cont
21
22 Output element
23 BeroepSector1.2
24
25 Aantal
26 &Cont 0 &Cont
27
28 Output element
29 BeroepSector2
30
31 o: CdSector o: OmsSector o: IndSectorActief
32 &Cont 0000031146 Gezondheidszorg, welzijn en persoonlijke verzorging 1 &Cont
33
34 Output element
35 BeroepSector2.1
36
```
The New Approach

Test Cluster and to generate the output TestFrame Report. Furthermore, the TestWare needs to be aware of the location of the various Web Services to be tested. This is solved by using a configuration file which contain the locations or endpoints of the Web Services to be tested.

Messaging

Messaging occurs by generating SOAP Templates with soapUI in the manner described in 3.

6.3 The New Approach

This section will cover the technical details of the approach using WSDL2TestFrame as described in chapter 5. This approach will be called the new approach. Again, the generic information present in chapter 3 is leading.

Test Clusters

Obviously, the new WSDL2TestFrame method requires Test Clusters to be in the WSDL2TestFrame Test Cluster format. Normally, templates for this format (generated by the WSDL2TestFrame Test Generation Phase) consequently manually populated with test data (by the WSDL2TestFrame Test Population Phase) resulting in the required Test Clusters. In this Case Study however, the test data has to be the same for both approaches for comparison purposes. Therefore, it is more efficient to reconstruct the Test Clusters from the original format to the new format. Specific for this reconstruction process, a python script has been written to automatically reconstruct the new Test Clusters based on the original Test Clusters. The script was able to do this by determining a global hierarchical structure in the meta-data present in the Test Cases, Action Words and Parameter Entries. This meta-data is not a required part of the actual Test Cluster format, but has been provided for testers and developers as an additional help. This pattern was not always present for each Action Word due to incorrect meta-data or inconsistent or ambiguous naming; in some cases the data could be manually corrected, in other cases the data could not. The cases in which the new Action Word could not be properly reconstructed, are apparent as such in the Test Results which are discussed in section 6.4.

Two subsets of the Action Words require some additional attention. A small subset of the Action Words in the Test Clusters relied on the test data present in a test repository. This subset consisted of five Action Words related to the getS-bcBeroepGroepForBeroep Web Operation. Using this subset would require additional implementation concerning databases, data(base) mapping and connectivity. As it is not the goal of the Case Study to test the aforementioned implementation, this subset will be present in the Test Results, but will not be included in the evaluation of the Case Study.
A different small subset of the Action Words test the length of certain data-structures/lists present in the Output Parameters. When manually analyzing this subset and the TestWare from the original approach, the subset seemed to be inconsistent; Test Cases similar in nature would test the lengths of different data-structures/lists for reasons that are unclear. Additionally, when performing a Test Run with the original TestWare, some expected test values are just plain incorrect. Most likely, the functionality of the original TestWare concerning this subset is incorrect. As more and more data are added to the system to be tested, the length of the data-structures/lists increase. Therefore, Action Words are constant out of date and will have to be constantly corrected. This correction is a cumbersome and non-productive process and has relative low priority. In general, incorrect Action Words cannot be used to verify the TestWare’s functionality. Therefore, it is suspected that the functionality of the TestWare concerning this subset has not been maintained and is possibly incorrect. In summary, because of the partly incorrect Action Words, the suspected partly incorrect functionality of the original TestWare concerning this subset and no other test documentation available, this subset cannot be adapted to the format of the new approach in a structured manner. As such, this subset as a whole has been excluded from the Case Study and thus are not present in the Test Results. The result of reconstructing the excerpt displayed in figure 6.1 is listed in 6.2. A simpler example is listed in appendix C.
The New Approach

Listing 6.2: An Excerpt from a converted Test Cluster in the Original Approach

```java
testcondition_boca01C10
    Get all active and inactive items of 'BeroepSector' with operation 'getAllBeroepSector'
    Not defined Not defined Must Test

testcase_boca01C10T10
    All active and inactive items of 'BeroepSector' have been found.  
    Not defined Not defined Must Test

getAllBeroepSector 12 &Cont
    Output element
    BeroepSector1
    o: BeroepSector[0].CdSector o: BeroepSector[0].OmsSector o: BeroepSector[0].
        IndSectorActief
    &Cont 0000311142 Automatisering en ICT 1 &Cont
    Output element
    BeroepSector1.1
    o: BeroepSector[0].BeroepSector[#]
    &Cont 3 &Cont
    o: BeroepSector[0].BeroepSector[0].CdSector o: BeroepSector[0].BeroepSector[0].
        OmsSector o: BeroepSector[0].BeroepSector[0].IndSectorActief
    &Cont 0000311143 Ontwerp en ontwikkeling 1 &Cont
    Output element
    BeroepSector1.2
    o: BeroepSector[0].BeroepSector[#]
    &Cont 0 &Cont
    Output element
    BeroepSector2
        IndSectorActief
    &Cont 0000311146 Gezondheidszorg, welzijn en persoonlijke verzorging 1 &Cont
    Output element
    BeroepSector2.1
```

Test Setup

The environment in which the tests took place is the same as in the original approach.
6. Case Study and Evaluation

Test Implementation and Execution

In general, any kind of TestWare is not needed at all, besides the already given WSDL2TestFrame, provided that no Web Operations with customized SOAP headers are to be tested. Web Operations with customized SOAP headers fall outside of this scope of this case study and are not supported by WSDL2TestFrame (yet). The GUI of WSDL2TestFrame has roughly the same interface as in the GUI in the original approach, except for the addition of a list of WSDL files and a console.

Test Execution occurs by performing Use Cases 2 as defined in 4.2. Again, every Test Cluster corresponds to a single Test Execution.

Messaging

Generating, populating, serializing, deserializing, and receiving of all the SOAP messages occur behind the scenes. These processes include generating, compiling and instantiating the relevant XMLBeans Classes as needed. Basically, the transport level is hidden from the outside within the WSDL2TestFrame method.

Test Results

The test results of the new approach are globally the same as the original approach. The main differences are: different Input/Output Parameters names which indicate the path of the relevant XML element, the additional check for null values retrieved from output XML elements. The test results are too large to be included in this document, but an excerpt from a Report of the new approach has been listed in figure 6.3.
Listing 6.3: An Excerpt of a Report of the New Approach

```csharp
1 testcondition  boca01C10
2 testcase  boca01C10T10
3
4 41 : getAllOpleidingSector 13 0000031142 Automatisering en ICT 1 3
5 0000031143 Ontwerp en ontwikkeling 1 3
6 0000031146 Gezondheidszorg, welzijn en persoonlijke verzorging 1 3
7 0000031147 Gezondheidszorg 1 0000031170 Handel en administratie 1 3
8 0000031171 Handel 1 0000031216 Horeca en huishouding 1 3
9 0000031217 Horeca 1 0000031224 Landbouw, natuur en milieu 1
10 0000031225 Planten 1

11 Check of : o: OpleidingSector[#]
12 Expected : 13
13 Recorded : 13
14 Result : passed

15 Check of : o: OpleidingSector[0].CdSector
16 Expected : 0000031142
17 Recorded : 0000031142
18 Result : passed

19 Check of : o: OpleidingSector[0].OmsSector
20 Expected : Automatisering en ICT
21 Recorded : Automatisering en ICT
22 Result : passed

23 Check of : o: OpleidingSector[0].IndSectorActief
24 Expected : 1
25 Recorded : 1
26 Result : passed

27 Check of : o: OpleidingSector[0].OpleidingSector[#]
28 Expected : 3
29 Recorded : 3
30 Result : passed

31 Check of : o: OpleidingSector[0].OpleidingSector[0].CdSector
32 Expected : 0000031143
33 Recorded : 0000031143
34 Result : passed

35 Check of : o: OpleidingSector[0].OpleidingSector[0].OmsSector
36 Expected : Ontwerp en ontwikkeling
37 Recorded : Ontwerp en ontwikkeling
38 Result : passed

39 Check of : o: OpleidingSector[0].OpleidingSector[0].IndSectorActief
40 Expected : 1
41 Recorded : 1
42 Result : passed

43
44
45
46
```

41
6. Case Study and Evaluation

Check of: o: OpleidingSector[1].CdSector
Expected: 0000031146
Recorded: 0000031146
Result: passed

Check of: o: OpleidingSector[1].OmsSector
Expected: Gezondheidszorg, welzijn en persoonlijke verzorging
Recorded: Gezondheidszorg, welzijn en persoonlijke verzorging
Result: passed

Check of: o: OpleidingSector[1].IndSectorActief
Expected: 1
Recorded: 1
Result: passed

Check of: o: OpleidingSector[1].OpleidingSector[#]
Expected: 3
Recorded: 3
Result: passed

Check of: o: OpleidingSector[1].OpleidingSector[0].CdSector
Expected: 0000031147
Recorded: 0000031147
Result: passed

Check of: o: OpleidingSector[1].OpleidingSector[0].OmsSector
Expected: Gezondheidszorg
Recorded: Gezondheidszorg
Result: passed

Check of: o: OpleidingSector[1].OpleidingSector[0].IndSectorActief
Expected: 1
Recorded: 1
Result: passed

6.4 Evaluation

The evaluation of the Case Study revolves around comparing multiple aspects of both approaches. Within the evaluation, the following aspects are highlighted: functionality and productivity. Evaluating the functionality is achieved by comparing the Test Results from both approaches. Evaluating the productivity is achieved by comparing the required manual intervention needed of both approaches.

6.4.1 Evaluating Functionality

To evaluate the functionality aspect the Test Results presented in the Test Reports from both approaches were compared. Within the Case Study, the Test Results are scattered among dozens of files, the Test Reports. For the sake of this aspect, a script has been developed to partially automate the process of comparing the Test
Evaluation

Results. The script parses all the Test Reports from both approaches, structures the data before storing in memory, compares the data and prints all the results. The script also contained some intelligence in order to recognize relevant trivial patterns in the Test Results. The script has been written in Python and roughly equals about 600 lines of source code. The (comparison) results of this script are too large to be included in this document, but an excerpt of the results have been listed in listing 6.4.
These results have been manually converted and summarized into a table format as can be seen in table 6.1 where ‘Tot’ stands for Total, ‘Mat’ for Matching, ‘NoM’
Evaluation

for Not Matching and ‘Mis’ for Missing. The values for Matching, Not Matching and Missing all amount to the Total value.

Table 6.1: The Comparison of the Test Results of Both Approaches

<table>
<thead>
<tr>
<th>Test Cluster</th>
<th>Tot</th>
<th>Mat</th>
<th>NoM</th>
<th>Mis</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Action Word</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>030</td>
<td>000</td>
<td>000</td>
<td>030</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T20</td>
<td>030</td>
<td>000</td>
<td>000</td>
<td>030</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T30</td>
<td>030</td>
<td>000</td>
<td>000</td>
<td>030</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T40</td>
<td>030</td>
<td>000</td>
<td>000</td>
<td>030</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T60</td>
<td>030</td>
<td>000</td>
<td>000</td>
<td>030</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T70</td>
<td>030</td>
<td>000</td>
<td>000</td>
<td>030</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T80</td>
<td>030</td>
<td>000</td>
<td>000</td>
<td>030</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T90</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T100</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T110</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T120</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T130</td>
<td>012</td>
<td>000</td>
<td>000</td>
<td>012</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T140</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T150</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
<tr>
<td>findBeroep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T160</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
</tbody>
</table>

45
6. Case Study and Evaluation

<table>
<thead>
<tr>
<th>Code</th>
<th>findBeroep</th>
<th>Beroep</th>
<th>findOpleidingsnaam</th>
<th>findOpleidingsnaam</th>
<th>getAllBeroepSector</th>
<th>getAllGedragsCompetentie</th>
<th>getAllGedragscompetentie</th>
</tr>
</thead>
<tbody>
<tr>
<td>boca01C10T170</td>
<td>030 000 000 030</td>
<td>Full Miss</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T180</td>
<td>030 000 000 030</td>
<td>Full Miss</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T190</td>
<td>030 000 000 030</td>
<td>Error</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T200</td>
<td>030 025 000 005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T210</td>
<td>006 000 000 006</td>
<td>Full Miss</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T10</td>
<td>000 000 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T20</td>
<td>000 000 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>findOpleidingsnaam</td>
<td>025 025 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>025 025 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T20</td>
<td>025 025 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T30</td>
<td>025 025 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T40</td>
<td>015 015 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T50</td>
<td>015 015 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T70</td>
<td>015 015 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T80</td>
<td>015 015 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T10</td>
<td>000 000 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T20</td>
<td>000 000 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T30</td>
<td>000 000 000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllBeroepSector</td>
<td>042 034 008 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllGedragsCompetentie</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>061 040 012 009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Code</td>
<td>Success</td>
<td>Reason</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------------</td>
<td>---------</td>
<td>---------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllOpleidingNiveauBemiddeling</td>
<td>boca01C10T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllOpleidingNivBemling</td>
<td></td>
<td>015 015</td>
<td>000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllOpleidingSector</td>
<td>boca01C10T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllOpleidingSector</td>
<td></td>
<td>030 030</td>
<td>000 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllSbcBeroepGroep</td>
<td>boca01C10T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllSbcBeroepGroep</td>
<td></td>
<td>030 016</td>
<td>008 006</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllSoiOpleidingRubriek</td>
<td>boca01C10T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllSoiOpleidingRubriek</td>
<td></td>
<td>030 014</td>
<td>016 000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td>boca01C10T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td></td>
<td>015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td>boca01C10T20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td></td>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td>boca01C10T30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td></td>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td>boca01C10T40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepDetails</td>
<td></td>
<td>005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td>boca01C10T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td></td>
<td>024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td>boca01C10T20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td></td>
<td>024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td>boca01C10T30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td></td>
<td>015</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td>boca01C10T40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td></td>
<td>024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td>boca01C20T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td></td>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td>boca01C10T50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepRelaties</td>
<td></td>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepSectorenForBeroep</td>
<td>boca01C10T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepSectorenForBeroep</td>
<td></td>
<td>023</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepSectorenForBeroep</td>
<td>boca01C10T20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepSectorenForBeroep</td>
<td></td>
<td>009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepSectorenForBeroep</td>
<td>boca01C10T30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepSectorenForBeroep</td>
<td></td>
<td>012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# 6. Case Study and Evaluation

<table>
<thead>
<tr>
<th>Function</th>
<th>ID</th>
<th>Status</th>
<th>Value1</th>
<th>Value2</th>
<th>Value3</th>
<th>Value4</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>getBeroepVerwantschappen</td>
<td>boca01C10T10</td>
<td>025</td>
<td>000</td>
<td>000</td>
<td>025</td>
<td>Full Miss</td>
<td></td>
</tr>
<tr>
<td>getBeroepVerwantschappen</td>
<td>boca01C10T20</td>
<td>025</td>
<td>000</td>
<td>000</td>
<td>025</td>
<td>Full Miss</td>
<td></td>
</tr>
<tr>
<td>getBeroepVerwantschappen</td>
<td>boca01C10T30</td>
<td>025</td>
<td>000</td>
<td>000</td>
<td>025</td>
<td>Full Miss</td>
<td></td>
</tr>
<tr>
<td>getBeroepVerwantschappen</td>
<td>boca01C10T40</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBeroepVerwantschappen</td>
<td>boca01C10T50</td>
<td>001</td>
<td>000</td>
<td>000</td>
<td>001</td>
<td>Full Miss</td>
<td></td>
</tr>
<tr>
<td>getBeroepsnamen</td>
<td>boca01C10T10</td>
<td>023</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getBeroepsnamen</td>
<td>boca01C10T20</td>
<td>001</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getBeroepsnamen</td>
<td>boca01C10T40</td>
<td>001</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getBeroepsnamen</td>
<td>boca01C10T50</td>
<td>001</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getBeroepsnamen</td>
<td>boca01C10T60</td>
<td>001</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T10</td>
<td>014</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T20</td>
<td>015</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T30</td>
<td>015</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T40</td>
<td>015</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T50</td>
<td>015</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T60</td>
<td>015</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T70</td>
<td>015</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T80</td>
<td>001</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>getOpleidingDetails</td>
<td>boca01C10T90</td>
<td>001</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Function</td>
<td>Code</td>
<td>Result</td>
<td>Validity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
<td>------</td>
<td>--------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T100</td>
<td>getOpleidingDetails</td>
<td>001</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>getOpleidingRelaties</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>getOpleidingRelaties</td>
<td>017</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T20</td>
<td>getOpleidingRelaties</td>
<td>001</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T30</td>
<td>getOpleidingRelaties</td>
<td>001</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T40</td>
<td>getOpleidingRelaties</td>
<td>001</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T50</td>
<td>getOpleidingRelaties</td>
<td>001</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>getOpleidingsnamen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T10</td>
<td>getOpleidingsnamen</td>
<td>010</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T20</td>
<td>getOpleidingsnamen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T30</td>
<td>getOpleidingsnamen</td>
<td>000</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T40</td>
<td>getOpleidingsnamen</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T50</td>
<td>getOpleidingsnamen</td>
<td>000</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>getRefOplForSoiOplRubriek</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T10</td>
<td>getRefOplForSoiOplRubriek</td>
<td>048</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T20</td>
<td>getRefOplForSoiOplRubriek</td>
<td>000</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C20T30</td>
<td>getRefOplForSoiOplRubriek</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>getReferentiebrnForSbcBeroepGroep</td>
<td>020</td>
<td>—</td>
<td>Invalid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T20</td>
<td>getSbcBeroepGroepForBeroep</td>
<td>006</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T30</td>
<td>getSbcBeroepGroepForBeroep</td>
<td>006</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
<td></td>
<td></td>
</tr>
<tr>
<td>boca01C10T10</td>
<td>getSbcBeroepGroepForBeroep</td>
<td>006</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Matching Value

The Matching value means that the new approach has the same Test Result as the original approach (for a particular Output Parameter). The Not Matching value means the opposite thereof. As it turns out, all the Not Matching cases are cases in which the Test Result from the original approach has passed, whereas the same Test Result of the new approach failed. In other words, it can be concluded that 1) the new approach did not generate any false positives other than the false positives which may or may not be already present in the original approach in this Case Study.

### Missing Value

The Missing value means that the new approach could not locate the listed Output Parameter in the SOAP output. A possible cause of this discrepancy is that the path in the Action Word is incorrect. The Action Words in which all of the listed Output Parameters could not be matched, are indicated by the remark Full Miss. Note that the Action Words in which no Output Parameters are present are not explicitly indicated as Full Miss in the table. The Action Words of three Web Operations, getBeroepVerwantschappen, getSbcBeroepGroepForBeroep and getSoiOpleidingRubriekForOpleiding are all denoted by a Full Miss. After some manual inspection of these specific cases, it has been concluded that the new Action Word could not be correctly reconstructed, as mentioned in section 6.3. This prevents the WSDL2TestFrame Application to retrieve the correct test data. As such, it is prudent to exclude these Web Operations when evaluating the functionality of the new approach. The affected Web Operations are listed at the top of section 6.1 with italic text.

<table>
<thead>
<tr>
<th>Web Operation</th>
<th>Test Result</th>
<th>Test Result</th>
<th>Test Result</th>
<th>Test Result</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSbcBeroepGroepForBeroep boca01C20T10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External</td>
</tr>
<tr>
<td>getSbcBeroepGroepForBeroep boca01C20T20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External</td>
</tr>
<tr>
<td>getSbcBeroepGroepForBeroep boca01C20T30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External</td>
</tr>
<tr>
<td>getSbcBeroepGroepForBeroep boca01C20T40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External</td>
</tr>
<tr>
<td>getSbcBeroepGroepForBeroep boca01C20T50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>External</td>
</tr>
<tr>
<td>getSoiOpleidingRubriekForOpleiding boca01C10T10</td>
<td>006</td>
<td>000</td>
<td>000</td>
<td>006</td>
<td>Full Miss</td>
</tr>
</tbody>
</table>

50
Evaluation

Invalid Remark

The Invalid remark means that Web Operation could not be successfully invoked due to an inconsistency with the WSDL document and the Web Operation; some Web Operations returned output SOAP messages different from those defined in the WSDL document. The differences in the Parameters were subtle, the names of the actual Output parameters were postfixed with a integer index which was unadvertised in the WSDL document. An example is the name ‘BeroepsnaamGecodeerdResponse’ as defined in the WSDL document versus the names ‘BeroepsnaamGecodeerdResponse1’ or ‘BeroepsnaamGecodeerdResponse2’ as seen in the various actual SOAP messages. This discrepancy is directly detected by Axis2, which results in a displayed error message regarding the relevant SOAP message in the WSDL2TestFrame console. As such, some of the Web Services could not be executed by the TestWare and have been excluded form the Case Study. The affected Web Operations are listed at the top of section 6.1 with strikethrough text. Although the WSDL document could be corrected manually so the Web Services could be executed, this has not been done due to time constraints.

After some investigation, it became clear that the TestWare of the original approach does not validate the SOAP messages against the WSDL document. Still, these discrepancies are defects in the Web Service. This is a direct example of how automated testing can detect these kind of errors. Errors which have not been observed in the original approach, but will consistently be observed in the new approach. In other words, II) the new approach can consistently and repeatedly detect inconsistencies between the WSDL document and the implementation.

Surprisingly, the old approach did contain seemingly sound Test Results related to aforementioned Web Operations. As it turns out, the TestWare of the old approach ignores and inappropriately overrides the WSDL document in the cases of aforementioned Web Operations. The fact that this behaviour conforms to the actual (inappropriate) implementation of the Web Service is of less importance. In other words, it can be concluded that III) from a strict functional point of view, this Case Study shows that, within the original approach, all the Test Results related to 9 out of 20 Web Operations are false positives. Nevertheless, these Web Operations are excluded from the evaluation of the functionality of the new approach. After excluding all the mentioned subsets of the Test Results, a vast amount of test data was available for the Case Study still.

Test Result Comparisons

After excluding the subsets mentioned in the previous sections from the evaluation, the exclusion of another Action Word subject to discussion. In one Action Word (boca01C10T190 of findBeroep) the WSDL2TestFrame Application generated an error and could not complete the testing process. This is the only error generated by the WSDL2TestFrame Application in the Case Study. The cause of this is currently unclear. Further investigation is required, but could not be performed due to time constraints. This Action Word concerns 30 Tests (Output Parameters) which
all are marked as Missing. Including this Action Word results in Test Results concerning 8 Web Operations containing 15 Test Cases and 15 Action Words with a Total of 403 Tests (Output Parameters) of which 309 were Matching, 44 were Not Matching and 50 were Missing. Excluding this Action Word results in Test Results concerning 8 Web Operations containing 14 Test Cases and 14 Action Words with a Total of 373 Tests (Output Parameters) of which 309 were Matching, 44 were Not Matching and 20 were Missing. Based on these quantifications, the relative accuracy of the new approach concerning this Case Study can be calculated. Note that this accuracy is the accuracy relative and proportional to the accuracy of the old approach. The actual (absolute) accuracy of the new approach may be higher than the old approach, because of the possible presence of false positives in the Test Results of the old approach (which are not present in the new approach). When calculating the accuracy, the accuracy is divided in two forms, Optimistic Accuracy, which excludes the Missing values of the test data and Pessimistic Accuracy which includes it. Also, the distinction has been made regarding the inclusion or exclusion of the erroneous Action Word. The results is listed in listing 6.2.

### Table 6.2: The Relative Calculated Accuracy Including and Excluding the Erroneous Action Word

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inclusive</td>
<td>8</td>
<td>15</td>
<td>15</td>
<td>403</td>
<td>309</td>
<td>268</td>
<td>41</td>
<td>44</td>
<td>50</td>
<td>88%</td>
<td>77%</td>
</tr>
<tr>
<td>Exclusive</td>
<td>8</td>
<td>14</td>
<td>14</td>
<td>373</td>
<td>309</td>
<td>268</td>
<td>41</td>
<td>44</td>
<td>20</td>
<td>88%</td>
<td>82%</td>
</tr>
</tbody>
</table>

**Test Result Interpretations**

It is clear that the Test Results of the new approach do not fully match the Test Results of the original approach. The comparison between the Test Results however, does not show which approach is more correct; listing 6.2 shows the calculated accuracy of the new approach relative to the original approach. This means that a relative accuracy less than 100% does not directly show that that approach was less accurate than the approach being compared with. In general, to be able to determine the accuracy of any approach, an oracle capable of providing true (expected) outcomes of tests would be required. In this Case Study, the expected values present in the original approach are used to be able to compare between both approaches, however these expected values are not truthful, as both approaches show that
various Action Words fail; i.e. the actual value does not match with the expected value according to both approaches. Also, a select group of failed Action Words have been manually verified (, i.e. the Action Words indeed are meant to fail). It is clear that the expected values cannot cannot represent an oracle. The fact that the expected values are not truthful would not pose a problem if an approach was available which would function correctly. However, the process of manually verifying a select group of contradicting Action Words, Action Words that are judged differently between both approaches, has shown that the TestWare of the original approach generated false positives in these select Action Words. Resolving all the false positives (or false negatives for that matter) of either approach would require manual and cumbersome labour which has not been performed due to time constraints.

However, it can be speculated that in any testing process, passed tests receive far less attention than failed tests and the testing process of the original approach should not be an exception. Failed tests or Action Words are associated with defects and will be shortly investigated upon, whereas passed Action Words are associated with correctness and will usually not be investigated upon. This is especially the case when regression tests are numerous. As such, for the sake of determining the accuracy of the new approach, failed Action Words are far more valuable than passed Action Words, as passed Action Words will contain more false positives than failed Action Words will contain false negatives, if at all. The total number of matched Failed Action Words of the original approach are 41, spread over 3 Web Operations and 6 Test Cases. These failed Action Words are fully matched by the new approach. In other words, it can be concluded that IV) the matched failed Action Words from the original approach, which are more reliable than the passed Action Words, are fully matched by the new approach. This conclusion is related to conclusion I).

6.4.2 Evaluating Productivity

To evaluate the productivity aspect the Process and Product Model of both approaches were compared and some user/developer feedback of the old approach was retrieved and interpreted. The obvious difference in the Product Model between the two approaches is the instance-specific TestWare Implementation versus the generic WSDL2TestFrame Application. Analyzing the generic nature of the WSDL2TestFrame Application shows some direct improvements on the Web Service testing process. As the WSDL2TestFrame Application is an implementation of the application described in the AWeSoMe method, the improvements of the WSDL2TestFrame Application should also apply to the AWeSoMe method.

Decrease of the TestWare Development Effort between Input/Output Parameters

When performing the Web Service testing process by the TestFrame method, there is a large number of recurring steps to be taken by the TestWare. The TestWare (together with the TestFrame Engine) has to read the Test Cluster token by token
and each token represents a Test Condition, Test Case and Action Word headers or Input or Output Parameter. Every Action Word requires some code to determine the correct Web Operation, to invoke the Web Operation. Every Input Parameter needs to be paired with the correct XML element and the XML element needs to be populated with the value of the Input Parameter and every Output Parameter needs to be paired with the correct XML element and the values of the XML element needs to be read and checked by the TestWare. Pairing the Input/Output Parameters, instantiating/populating the XML element and reading and checking the values of XML elements all require some lines of code. In the old approach, this code needs to be manually implemented, which means that development effort spent on the TestWare grows at least linearly with every added Action Word or Input/Output Parameter. This is different in the new approach, as WSDL2TestFrame generates all of the aforementioned code. Thus, after the initial (constant) effort of developing the WSDL2TestFrame, no further development is spent on the aforementioned actions needed when reading a single token from the Test Cluster. In other words, it can be concluded that: V) with the new approach, the development effort decreases linearly with every additional Input/Output Parameter of the WSUT.

Decrease of the TestWare Development Effort between Subsequent Modifications of the WSUT

It is not uncommon that during the development process the implementation starts to differ from the specifications. This has been acknowledged by user/developer feedback of the old approach. The same holds for the implementation or the WSUT and the specified Test Cases. This effect is amplified as the WSUT and Test Cases are often developed in parallel, usually from the very start. These differences affect the mapping between the Test Cases and the Web Operations of the WSUT. Modifying a Web Operation can be categorized in the following cases:

1. The name of the Web Operation does not match (anymore).
2. The name of an Action Word parameter does not match (anymore).
3. The position/path of an Action Word parameter does not match (anymore).
4. The number of Action Word parameters does not match (anymore).
5. The one-to-one mapping of Action Word-Web Operation has changed to a multiple-to-one mapping.
6. The one-to-one mapping of Action Word-Web Operation has changed to a one-to-multiple mapping.

These cases render the mappings incorrect which will result in false positives/negatives in the Test Results or (run-time) errors in the tests. In the original approach, the discrepancies have to be resolved by either revising the Action Words
or by revising the TestWare. After every modification, the mappings with the Web Operations of the WSUT have to be checked and corrected if necessary. Naturally, any following modification of the Web Operations in the development process can cause new incorrect mappings. Case 1 and 2 should be trivial to resolve. Case 3 and 4 require more effort. Case 6 is problematic, as there is no Action Word format which specifies that multiple Web Operations should be invoked. Resolving Case 6 means redefining the Action Word format itself. Case 5 is out of scope of this Case Study. Case 5 is not a problem as any Web Operation can be invoked multiple times.

The new approach has better support for these unstable mappings. As the Test Cluster format of the new approach enforces that the structure or path is embedded in the name of the Input/Output Parameter and as this path is utilized by WSDL2TestFrame to correctly and unambiguously map the Input/Output Parameter to the correct XML elements within (SOAP messages), any additional revising of the implementation for the sake of this mapping is no longer needed. In other words, after the initial effort of implementing WSDL2TestFrame itself, the TestWare does not need to be revised any further to support the mapping of any (revised) Input/Output parameter of any testing process. Any change of the Test Cases are restricted to the Test Cases themselves and do not propagate to the TestWare.

Of course, when initially defining an Input/Output parameter, some care should be taken to correctly include the name of the desired Input/Output parameter, however this should be a trivial matter as WSDL2TestFrame generates an example Test Cluster exhibiting the correct names which can easily be duplicated. Should the Web Service and its WSDL document change, the example Test Cluster can readily be regenerated with the updated naming. In other words, it can be concluded that:

VI) with the new approach, the development effort decreases linearly with certain types of subsequent modified Web Operations of the WSUT compared to the old approach.

Decrease of the Error-Prone Factor

The generic nature of the WSDL2TestFrame Application and the new Test Cluster format with its embedded structure or path of Input/Output Parameters makes generating or revising Test Clusters in the new approach a trivial task. The example Test Cluster generated by WSDL2TestFrame Application already contains most if not all of the needed paths of the Input/Output parameters. Only thing left for completing a Test Cluster is inserting the desired test values. Additionally, the use of soapUI and SOAP templates with its inserted values are not needed anymore, reducing the error-prone factor even more. In other words, it can be concluded that:

VII) with the new approach, the process of revising the test data Artifacts during the testing process is less error-prone than in the original approach.

6.5 Threats to Validity

This section discusses the threats to validity of this Thesis.
6. Case Study and Evaluation

Domain Deprivation

First and foremost, the Thesis and its Case Study is situated in one domain only. It is unclear if the outcome of this Thesis is domain-independent or if the outcome is only applicable to a set of domains of for this domain only.

Web Service and Web Operations Deprivation

The Case Study involves one Web Service only and within the Web Service only 8 out of the total of 20 Web Operations were included. It is unclear if the outcome of the Case Study will be same with other Web Services. The Case Study excluded the Web Operations which could not be executed due to discrepancies between the WSDL document and its implementation. It is possible that the excluded Web Operations (and its Action Words) would show a different outcome than what the current Case Study shows.

Action Word Deprivation

The Case Study excludes Action Words which could not be properly formatted in the new format employed by the new approach. The formatting process was based on additionally provided meta-data in the original Test Cluster which was not always complete, consistent or unambiguous. It is possible that the excluded Action Words would show a different outcome than what the current Case Study shows.

False Positives and Negatives

False positives and negatives present in the output of various implementations are a threat to validity. Multiple implementations were used in the Case Study; the original TestWare, the (Test Cluster) reconstruction script, the WSDL2TestFrame Application and the Test Report parser script. False positives and negatives, if present, could effect any stage within the Case Study and the Thesis and could negatively affect the conclusions of this Thesis.
Chapter 7

Related Work

This chapter lists the approaches found in the academic literature which already have been undertaken regarding automated testing of Web Services.

7.1 Web Service Testing by Parameter Variance

The testing of Web Services resembles the testing of interfaces which is a form of black-box testing [23]. WSDL defines in detail the interface of a particular Web Service. The functional design may not be available when testing Web Services, for example when service consumers are not the same party as service providers. In any case, disregarding the functional design during Web Service testing equals to running numerous tests on the Web Service with subsequent different parameters; i.e. Web Service Testing by varying the parameters of Web Operations. Salva and Rabhi [33] call this kind of testing Web Service Robustness Testing. Varying the parameters of Web Operations leads to a large number of test cases; the generation of test cases is not a small task. To limit the number of test cases, it is helpful to know how parameter variation determine this robustness of Web Services. [33] lists the following hazards relevant to parameter variance and Web Service testing:

- **Replacing parameter types** - replacing one or more parameter types with other types
- **Adding/injecting parameter types** - adding parameter types at the beginning of the request or between existing parameters
- **Deleting parameter types** - deleting parameters at the beginning or between existing parameters
- **Inverting parameter types** - comparable to Replacing parameter types
- **Calling with unusual values** - calling the operation with unusual predefined values, like for example NULL, "", "$", "*" are some unusual String values.
Salva and Rabhi [33] divide the implementation of a Web Service in two different entities: the generic SOAP processor and the actual Web Service operation implementations. As the first four listed types of hazards are not accepted by the SOAP processor, only the last type of hazard, Calling with unusual values, can be used to test the Web Service implementation.

Salva and Rabhi [33] place the emphasis on whether (an operation) a Web Service is robust or not. Determining the robustness of a Web Service means distinguishing between the SOAP processor fault/error handling and the actual Web Service operation implementation. SOAP processors can affect test results; the SOAP processor may generate SOAP faults even if the Web Service has crashed, for example, by throwing an exception. When executing test cases and inspecting test results, Salva and Rabhi [33] use the following heuristic to determine Web Service operation robustness:

- If the response is not a SOAP fault and if its type is the one described in the WSDL file, the local verdict is "pass".
- If the response is a SOAP fault whose cause is equal to "RemoteException" then the operation manages exceptions by itself and the local verdict is "pass".
- If the response is a SOAP fault whose cause is not in "RemoteException", "client", "the endpoint reference not found", then the operation exists but does not manage exceptions by itself. In this case, the local verdict is "inconclusive".
- Otherwise, the local verdict is "fail".

The WS-I is aware of the difference between SOAP processor faults/errors and Web Service implementation fault/errors and prescribes different behavior for these two categories in order to distinguish these two categories. These guidelines are formulated in the WS-I Basic Profile.

7.1.1 Parameter Variance as Data Perturbation

As can be seen in the previous section, omitting/removing defined parameters, adding undefined parameters or modifying the defined type of the parameters does not test the actual Web Service operation implementations, but rather the generic SOAP processor. To test the actual Web Service operation implementation, only the variance of values of defined parameters can be applied. This process involves exploring the (value) range of parameters. Offutt and Xu [24] call this parameter variation ‘Data Perturbation’. Within Data Perturbation, Offutt and Xu [24] make the distinction between Interaction Perturbation and Data Value Perturbation (DVP) and within Interaction Perturbation, the distinction has been made between RPC Communication Perturbation (RCP, where RPC stands for Remote Procedure Call) and Data Communication Perturbation (DCP):
• **RPC Communication Perturbation**
  Within RCP, a Web Service is viewed as a RPC interface. This originates from the practice of wrapping existing RPC interfaces within Web Services. A consequence of this, is that the structure of data elements is not seen as a tree hierarchy, such as is the case with XML, but rather as a flat hierarchy. The focus lies on testing the use of data. Offutt and Xu[24] distinguish two uses, normal data uses and Structured Query Language(SQL) uses. Normal data values are perturbed by randomizing operators and SQL data values are based on fault-generation by SQL injection. The latter assumes that knowledge of the implementation details regarding database querying is available.

• **Data Communication Perturbation**
  Within DCP, the focus lies on database relationships and constraints. The structure of the data elements is not viewed as a flat hierarchy, but rather as a relational database structure with relationships via primary and foreign keys. Constraints in XML schema, such as minOccurs and maxOccurs are mapped onto database constraints, such as: null not allowed and non-null uniqueness. On this basis test cases are generated. This approach assumes that knowledge of the database design is available.

• **Data Value Perturbation**
  Within DVP, values of SOAP messages are modified according to rules defined on the primitive data types of the XML values, by way of boundary value testing. Tests are created by replacing each value with boundary values; some examples of this being maximum length, upper case and 0. Unfortunately, it is not clear whether DVP is also based on XML constraints mentioned in DCP, such as minOccurs and maxOccurs.

Where possible, XML Schemas are modeled by Regular Tree Grammars(RTG). Test cases are generated by means of pre-defined operators operating on these RTG models. This formal model has been described in [24] and [43].

In [24], an empirical study was performed, where 100 DVP tests, 15 RCP tests, and 27 DCP tests, were generated. Of the total of 18 inserted faults in a particular SOA system, RCP tests found 7, DCP tests found 4 faults and DVP tests found the aforementioned faults and more, for a total of 14 faults.

### 7.1.2 Automation of Web Service Testing

As seen in [33], the test cases are generated by utilizing predefined data values. Also, forms of automated test execution is not discussed. [2] on the contrary, mentions a number of applications capable of offering some form of automation regarding the testing process, namely soapUI [1], SoaTest [30] and TestMaker [31] and evaluates the soapUI application in particular.
soapUI

soapUI is an application developed by Eviware Software AB, available both in free and improved commercial versions, assisting in the testing process of SOAP-based systems. It has the following features:

- generate stubs or templates of SOAP messages calls based on (the operations in) WSDL documents.
- sends SOAP messages to the web service and display the outputs
- mock a Web Service by providing a listening server compliant with a WSDL document
- populate a data source and generate messages with data extracted from it (only in the commercial version)
- run batch tests, and (in the commercial version) produce some partial information about coverage of the data value sets used in the operations

An example of such a SOAP message template of a Web Operation from the Case Study in chapter 6 is displayed in listing 3.1. However, the SOAP message templates have the following shortcomings:

- Skeleton SOAP messages include all the elements of the XML Schema, requiring user intervention and domain knowledge to delete parts of the structure in order to construct valid and usable test cases. Notable examples are ChoiceType or AllElem elements.
- Data values are mostly predetermined, or generated for Simple Types in a general fashion (in the commercial version), and thus not based on input specifications in the WSDL. Additionally, the dynamic nature of possible occurrences (occurElem) of elements is not fully utilized.

The application ‘Testing by Automatically generated XML Instances’((WS-)TAXI) aims to address these shortcomings [2].

WS-TAXI

Assuming, WS-TAXI utilizes the soapUI application and uses its output to further refine the skeleton SOAP messages and transform them into ready-to-launch test cases. To accomplish this, WS-TAXI applies the the Category Partition(CP) technique [29] on the XML Schema. The CP technique is used to achieve minimal sets of test cases. Additionally, the XML Attributes minOccurs and maxOccurs are used as boundary values in the generation of test cases. Data values are inserted by random generation or read from a database.

Bartolini et al.[2] mention the following limitations of WS-TAXI:
• WS-TAXI does not provide a complete test oracle, other than that all SOAP messages should be accepted by the SUT. Instead it provides the test results to the user and supposedly checks for user annotations.

• WS-TAXI does not consider possible dependencies between messages, however this is a limitation of syntax-based testing methods.

WebSob

[23] discusses a framework called WebSob for robustness testing of Web Services by way of client code generation. WebSob consists of the following components [23]:

• Code Generation Component
  Web Services to be tested are wrapped into Java classes, generated by the Axis utility class WSDL2Java; the Java classes contain the Web Service operation calls.

• Test Generation Component
  Test suites are created based on the wrapper Java classes; the Java classes are fed into unit test generation tools such as JCrasher, Agitar Agitator and Parasoft JTest. Generated data values are based on randomization or on basic boundary/extreme values. Assumingly, the data values are not dynamically generated from the WSDL specification.

• Test Execution Component
  Execution of the generated test suites.

• Response Analysis Component
  Collection of the test results, utilizing the Axis utility TCPMonitor. Test cases that show potential robustness problems are presented to the user.

7.2 Web Service Test Scheduling

As the implementation of Web Services is not publicly available, it is not explicitly obvious for a Service Consumer when implementations of Web Services change. This is illustrated by Bruno et al.[3] by mentioning two differences between SOA and traditional component based software:

• Addition of New Features
  Services may benefit from additional features, however, the service provider may choose not to advertise such change, as the service definition, including input and output parameters, is not affected. The introduced change possibly alters service behavior or non-functional properties, such as response time.
7. Related Work

• **Addition of Optimizations**
  Optimizations cause a variation in the service’s non-functional properties. Optimizations could improve one non-functional property while worsening another. In addition, an improvements of non-functional properties may not be desirable since that may cause undesired side effects.

Of course any change could introduce faults which has an influence on service functional behavior as well and any change may lead to a violation of the Service Level Agreement (SLA) stipulated between service consumer and service provider. Because of this, it might prove useful to test Web Services on a periodic basis, to determine if an implementation of Web Service has changed.

7.2.1 Periodic Testing of Web Services

Bruno et al.[3] propose to support service consistency verification through evolution by coupling a service with:

1. A set of test cases
2. A set of Quality of Services (QoS) assertions

These test cases and assertions constitute a kind of ‘contract’ between service consumer and service provider. By executing the test cases, the service consumer can observe and assess the service’s functional and non-functional properties. Based on this assessment, service consumer and provider are able to stipulate the contract, for a specified period of time.

The service consumer can periodically re-execute the test suite against the service to verify whether the service still exhibits the functional and non-functional properties when the service was acquired. Deviations from the agreed properties, by modifying the service, would cause a contract violation by on the part of the service provider.

In concrete, Bruno et al.[3] realize this approach by introducing XML-encoded test cases, called an XML-encoded facet, and QoS assertions. The XML-encoded facet contains test cases defined by the service provider and are executable by the service consumer. After an execution of any number of test cases by the service consumer the QoS can be asserted. This approach occurs as follows:

• **Generating the XML-encoded test suite**
  Bruno et al.[3] have developed a XML-encoded facet generator, which automatically generates a XML-encoded facet from a JUnit test case and the service class. Executing the test cases reveals the QoS properties. Response time and throughput were measured, but with more advanced monitoring systems, more complex QoS properties can also be measured.
Running the test suite
After publication of the service and the test suite, service consumers can run the test suite and monitor the QoS. Service consumers can also provide and publish further test cases, thus supplementing the already present test suite. These test cases can also reflect the intended use of the service in general.

The whole approach is supported by a toolkit, which is developed in Java and consists of two modules:

- The testing facet generator
  A tool generating the \textit{XML-encoded testing facet}. Currently, the tool accepts JUnit2 test suites, although test suites developed using other tools (and for services written using different languages) should also be possible. JUnit supports the development of a unit test suite as a Java class, containing a series of methods that constitute the test cases. Each test case is composed of a sequence of assertions checking various properties of the class under test. The tool relies on JavaCC 3 to perform Java source code analysis and transformation, on the Axis web services framework\textsuperscript{4} on the Xerces\textsuperscript{5} XML parser.

- The test suite runner
  A utility monitoring the execution of the test suite.

Although the focus lies on consumer-side testing or testing from the consumer perspective, there are also other testing perspectives:

- Provider/developer perspective QoS assessment for internal use by developers
- Consumer perspective Self QoS assessment by the consumer
- Certifier perspective An independent QoS assessment on behalf of the consumer

7.2.2 Self-Checking
Canfora and Di Penta\cite{5} mention an alternative to testing, namely continuous self-checking by means of monitoring a SOA system during execution. The advantages of self-checking over testing are:

- self-checking, being performed at run-time, also takes the intrinsic dynamicity and adaptability of SOA into account
- self-checking has minimal run-time overhead and produces no undesired side-effects, as every service invocation has actual real-world use

On the other hand, there are some disadvantages:
7. Related Work

Figure 7.1: Overview of ongoing monitor of SOA systems

- self-checking, most of the time, will not cover exceptional conditions, such as peak usage or seldom used functionality
- self-checking can only detect problems during run-time, which is too late to resolve for those particular executions

Thus, as Canfora and Di Penta[5] point out that testing and self-checking are two complementary facets. Testing is necessary to gain insights in the correctness of service-oriented systems before system deployment and in contrast, self-checking is necessary to assess and ensure the correctness of these dynamically changing systems after system deployment.
Chapter 8

Conclusions and Future Work

8.1 Contributions and Conclusions

The contributions of this Thesis are the following:

1) The result of the analysis of the current approach of applying the TestFrame Method on Web Services in terms of products and processes

The current approach of applying of the TestFrame Method on Web Services has been described by differentiating between the Product Model and Process Model and determining the various processes, artifacts/products and the relations between them. It has been observed that the application of the TestFrame method on Web Services in practice differed from the theoretical TestFrame method described in [35] and the discrepancies between them have been determined.

2) The definition of the AWeSoMe method which defines various forms of automation of the Web Service testing process

The AWeSoMe method has been defined which defines various forms of automation and subsequently has the potential to decrease spent effort and time of the Web Service testing process in general. This method is a generic method and is separate from various technologies, domains or contexts other than Web Services.

3) The result of the analysis of the new approach of applying the TestFrame Method on Web Services in terms of products and processes

The AWeSoMe method combined with the TestFrame method and Web Services, generates a new approach to apply the TestFrame method on Web Services. This new approach has also been described by differentiating between Product Model and Process Model and determining the various processes, artifacts/products and the relations between them. This will greatly enhance the ability to compare the two approaches.
4/5) Performing a Case Study of the original and the new approach assessing the validity, reliability and correctness of the new approach compared to the original approach. This includes developing a proof-of-concept WSDL2TestFrame Implementation, formatting test data and formatting, interpreting and evaluating the test results.

A Case Study has been performed involving the original approach and the new approach in the context the BOCPort Web Service. The feasibility of the AWeSoMe method and the new approach has been demonstrated by the WSDL2TestFrame Application. The application has been developed in Java. The metric Lines Of Code (LOC) of the application has been determined by the Eclipse Metrics Plugin [34], which resulted in 4740 LOC. The metric LOC excludes empty lines, comments and Javadocs. The total lines of source code, including empty lines and comments are estimated at 5500 lines. The AWeSoMe (Automatic Web Service Mapping) method and the WSDL2TestFrame implementation thereof has made the following possible:

- Automatic generation of AWeSoMe Test Templates including AWeSoMe Tests and AWeSoMe Parameters
- Automatic handling of the Input Parameters of AWeSoMe Tests of Web Services
- Automatic Test Execution of Web Services based on AWeSoMe Tests of Web Services
- Automatic handling of the Output Parameters of AWeSoMe Tests of Web Services
- Automatic generation of the AWeSoMe Test Results of AWeSoMe Tests of Web Services

Within the Case Study the functionality and productivity aspects of the new approach compared to the original approach have been evaluated. Evaluating the functionality is achieved by comparing the Test Results from both approaches. Based on the comparisons of the Test Results of the original and the new approach in the Case Study, the accuracy of both approaches could be measured against each other. The lack of an oracle providing the true (expected) outcomes of tests implies that both approaches could only compared relatively to each other and thus it cannot be concluded that one approach was more accurate than the the other. However, the validity of the Test Results of the Case Study have been maximized by focussing on only the Passed Action Words, instead of the Failed Action Words, as the chance of occurring false negatives is smaller than the chance of occurring false positives. When evaluating the functionality, the following has been concluded:

I) the new approach did not generate any false positives other than the false positives which may or may not be already present in the original approach in this Case Study

II) the new approach can consistently and repeatedly detect inconsistencies between the WSDL document and the implementation
III) from a strict functional point of view, this Case Study shows that, within the original approach, all the Test Results related to 9 out of 20 Web Operations are false positives.

Evaluating the productivity is achieved by comparing the required manual intervention needed of both approaches. When evaluating the productivity, the following has been concluded:

IV) the matched failed Action Words from the original approach, which are more reliable than the passed Action Words, are fully matched by the new approach.

V) with the new approach, the development effort decreases linearly with every additional Input/Output Parameter of the WSUT.

VI) with the new approach, the development effort decreases linearly with certain types of subsequent modified Web Operations of the WSUT compared to the old approach.

VII) with the new approach, the process of revising the test data Artifacts during the testing process is less error-prone than in the original approach.

Currently, it is still unclear if the outcome of this Thesis and its Case Study is domain-independent or if the outcome is only applicable to a set of domains or for this domain only. It is also unclear if the outcome of the Case Study will still be the same with the excluded Web Operations or with other Web Services. As always, False positives and negatives present in the output of various implementations are a threat to validity. Multiple implementations were used in the Case Study; the original TestWare, the (Test Cluster) reconstruction script, the WSDL2TestFrame Application and the Test Report parser script. False positives and negatives, if present, could affect any stage within the Case Study and the Thesis and could negatively affect the conclusions of this Thesis.

8.2 Future work

This section contains various courses of action which can be undertaken in order to continue the effort related to this subject:

8.2.1 Automated Test Case Generation and Execution Based on the Properties of Parameters

As can be seen in [23] and [2], it is possible to define Test Cases and test data of Web Services in an automated fashion. Generated test data can be based on randomization or on basic boundary/extreme values[23] or by XML Attributes specified in the WSDL document of Web Services such as minOccurs and maxOccurs which represent boundary values of Input/Output Parameters[2]. This was meant to be one of the ambitious features of WSDL2TestFrame, but could not be implemented due to time constraints. WSDL2TestFrame currently only generates a single constant value for each XML Type (i.e. string, double, etc.) for each Input/Output Parameter.
8. Conclusions and Future Work

8.2.2 Consulting Test Data Repositories

The processing of test data does not have to be limited to the Test Cluster format; test data repositories can also be consulted. A small subset of the test data of the original approach of this case study was already retrieved from a test data repository. Retrieving test data from a repository has also been applied in [2]. The original approach however, contained hard-coded logic determining when test data should be consulted externally. An extension of the Test Cluster format with grammar indicating this process would be preferred. A database is the most obvious form of a data repository, but other forms are possible.

8.2.3 Automated Test Case Generation and Execution Based on Best Practices

As mentioned earlier, WSDL and/or UDDI define interfaces of Web Services, without any further specifications. The Web Services Interoperability Organization (WS-I) argues that Web Services or WSDL suffers from underspecification [28][27]; some specifications of WSDL and UDDI are sometimes too general, too ambiguous, or lacking. Of course, since WSDL and UDDI are only definitions, this does not result from a shortcoming of WSDL or UDDI itself. Nevertheless, it is necessary to establish ‘Best Practices’ (of Web Services) such that the interoperability of Web Services can be as forthcoming as possible, at least, according to the WS-I. WS-I has published some tooling capable of aiding the testing process of the interoperability of Web Services. To enforce these ‘Best Practices’, a potential course of action would be to add automated Test Case generation based on these ‘Best Practices’ to WSDL2TestFrame.

8.2.4 Automated Test Case Generation and Execution Based on Semantics, Business Processes or Workflows

WS-I is not the only organization which made an effort to cope with the aforementioned underspecification of Web Services and/or WSDL. It has been suggested to extend WSDL itself with annotations specifying behavior of Web Services [37] which will result in additional testing possibilities. However, the most effort has been made into integrating Web Services and/or WSDL with external languages concerning semantics [12][40][4] or Business Processes/Workflows [26][42]. A potential course of action would be to add automated Test Case generation based on these behavioral specifications to WSDL2TestFrame.
Bibliography


Appendix A

The Technical Diagrams of the WSDL2TestFrame Implementation
Figure A.1: The first set of the Package Diagrams
Figure A.2: The second set of the Package Diagrams
Figure A.3: The overview Class Diagram
Figure A.4: The high-layer Class Diagram
A. The Technical Diagrams of the WSDL2TestFrame Implementation

Figure A.5: The low-layer Class Diagram
### WSDL2TestFrame

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL2TestFrame()</td>
<td>Constructor</td>
</tr>
<tr>
<td>initComponents()</td>
<td>Initializes components</td>
</tr>
<tr>
<td>actionState()</td>
<td>Handles ChangeEvent</td>
</tr>
<tr>
<td>actionPerform()</td>
<td>Handles ActionEvent</td>
</tr>
<tr>
<td>jList1ValueChanged()</td>
<td>Handles ListSelectionEvent</td>
</tr>
<tr>
<td>actionCluster1()</td>
<td>Handles ActionEvent</td>
</tr>
<tr>
<td>actionWSDL1()</td>
<td>Handles ActionEvent</td>
</tr>
<tr>
<td>actionWSDL2()</td>
<td>Handles ActionEvent</td>
</tr>
<tr>
<td>init()</td>
<td>Initializes the frame</td>
</tr>
<tr>
<td>refreshWSDLList()</td>
<td>Refreshes WSDL list</td>
</tr>
<tr>
<td>refreshClusterList()</td>
<td>Refreshes Cluster list</td>
</tr>
<tr>
<td>refreshStubList()</td>
<td></td>
</tr>
<tr>
<td>selectInWSDLList()</td>
<td>Selects in WSDL list</td>
</tr>
<tr>
<td>selectInClusterList()</td>
<td>Selects in Cluster list</td>
</tr>
<tr>
<td>selectInList()</td>
<td>Selects in List</td>
</tr>
<tr>
<td>listFiles()</td>
<td>Returns list of files</td>
</tr>
<tr>
<td>readWSDL()</td>
<td>Reads WSDL file</td>
</tr>
<tr>
<td>readCluster()</td>
<td>Reads Cluster file</td>
</tr>
<tr>
<td>fromWSDLFileToClusterFile()</td>
<td>Converts WSDL to Cluster file</td>
</tr>
<tr>
<td>fromClusterToReportFile()</td>
<td>Converts Cluster to Report file</td>
</tr>
<tr>
<td>main()</td>
<td>Main method</td>
</tr>
</tbody>
</table>

Figure A.6: The WSDL2TestFrame Class
SOAGateway

SOAGateway(in wsdlFile: String, in clusterFile: String, in reportFile: String, in serviceName: String, in os: OutputStream)

- getServiceNameFromCluster(): String
- axis2Loggingstart(): boolean
- axis2LoggingEnd():
- processWSDL(): boolean
- processCluster(): boolean
- compileBeans():
- compileAxis2(in compilerOutput: String, in elem: String): String
- handleAxis2Emitter(in reader: BufferedReader): String
- matchAxis2(in line: String, in elem: String): String
- reflectAndWrite(in stubName: String, in handlerName: String, in list boolean): boolean
- reflectAndRead(in stubName: String, in handlerName: String, in clusterFile: String, in reportFile: String, in iniFile: String): boolean
- update(in a: Observable, in arg: Object)
- flush():
- toSysErr(in e: Throwable)
- toSysErr(in line: String)
- toSysOut(in line: String)
- isValidToGenerate(): boolean
- isValidToRun(): boolean
- createWriter(in clusterFile: String): Writer
- generateUrlList(): URL[]
Figure A.8: The XmlBeansHandler Class
The Technical Diagrams of the WSDL2TestFrame Implementation

Figure A.9: The XmlBeansClusterReader Class

```java
XmlBeansClusterReader(stubName: String, inHandlerName: String, inURL: URL[], inClusterFile: String, inReportFile: String, inInFile: String)
  uri()
  handleTestCluster()
  handleTestCondition(): boolean
  handleTestCase(): boolean
  handleActionWord(entry: String, String>)
  printActionWordStart(entry: String, String>)
  printActionWordEnd(entry: String, String>)
  handleWebOperation(operation: Method)
  handleInputDocument(classDocument: Class<? extends XmlObject>?, XmlObject)
  handleOutputDocument(classDocument: Class<? extends XmlObject>?, inXmlDocument: XmlObject)
  handleWebOperationInputParameters(inParentNode: XmlBeansNode)
  handleWebOperationOutputParameters(inParentNode: XmlBeansNode)
```
Figure A.11: The XmlBeansDocument Class

```java
XmlBeansDocument(in clazz: Class<? extends XmlObject>, in loader: URLClassLoader)
XmlBeansDocument(in clazz: Class<? extends XmlObject>, in xmlObject: XmlObject, in loader: URLClassLoader, in spawn: boolean)
map()
- setContents()
- getOrSetParameterEntry(in type: ParameterType, in entry: ParameterEntry): String
- getOrSetUninListChild(in type: ParameterType, in entry: ParameterEntry): String
- getOrSetNodeChild(in type: ParameterType, in entry: ParameterEntry): String
- getOrSetNodeUninListChild(in type: ParameterType, in entry: ParameterEntry): String
- normalizeThrowables(in entry: ParameterEntry, in: t: Throwable)
- getUnInitMethodMap(): TreeMap<String, TreeMap<ParameterType,Method>>
- getUnInitListMethodMap(): TreeMap<String, TreeMap<ParameterType,Method>>
- getUnNodeMap(): TreeMap<String, XmlBeansNode>
- getNodeList(): TreeMap<String, TreeMap<ParameterType,Method>>
- getNodeListMap(): TreeMap<String, ArrayList<XmlBeansNode>>
- getNodeListMethodMap(): TreeMap<String, TreeMap<ParameterType,Method>>
- getNodeClass(): Class<? extends XmlObject>
- getNodeObject(): XmlObject
- XmlDocumentObject(): XmlObject
- XmleBeansDocument(in clazz: XmlObject, in xmlObject: XmlObject, in loader: URLClassLoader)
```
Figure A.12: XmlBeansNode Class Part 1

```
xmlelement

XmlBeansNode(in parent: XmlBeansNode, in clazz: Class<? extends Object>, in loader: URLClassLoader)
xmlelement

xmlelement

getParameterEntry(in entry: ParameterEntry): String
xmlelement

setParameterEntry(in entry: ParameterEntry): String
xmlelement

getOrSetParameterEntry(in type: ParameterType, in entry: ParameterEntry): String
xmlelement

getUnitTestChild(in entry: ParameterEntry): String
xmlelement

getUnitTestListChild(in entry: ParameterEntry): String
xmlelement

setUnitTestListChild(in entry: ParameterEntry)
xmlelement

generateGenericArray(in clazz: Class<T>, in length: int): T
xmlelement

XmlNodeChild(in entry: ParameterEntry): String
xmlelement

XmlNodeChild(in entry: ParameterEntry)
xmlelement

XmlNodeListChild(in entry: ParameterEntry): String
xmlelement

XmlNodeListChild(in entry: ParameterEntry)
xmlelement

XmlNodeListChild(in entry: ParameterEntry)
xmlelement

concreteListNode(in type: ParameterType, in listName: String): XmlBeansNode
xmlelement

map()
xmlelement

mapUnits(in method: Method)
xmlelement

mapUnitLists(in method: Method)
xmlelement

mapNodes(in method: Method)
xmlelement

mapNodeLists(in method: Method)
xmlelement

matchGetUnitMethod(in method: Method): String
xmlelement

matchSetUnitMethod(in method: Method): String
xmlelement

xmlelement

xmlelement

matchGetUnitListMethod(in method: Method): String
xmlelement

matchSetUnitListMethod(in method: Method): String
xmlelement

xmlelement

xmlelement

matchGetSingletonClass(in method: Method, in prefix: String): Class<?>
xmlelement
```
Appendix B

Background Information of the Considered Implementations

This appendix contains the relevant background information of the considered implementations which contain overlapping aspects with the WSDL2TestFrame Application. Because it is in the interest of Logica to retain the right on how to publish the WSDL2TestFrame Application, this restricts us to software under the Apache License, Mozilla License, Lesser General Public license or any other similar license. With this in mind, the following implementations have been found:

- The Apache Axis and Axis2 framework
- The java.net JAX-WS Reference Implementation
- The Apache CXF framework
- WS-TAXI
- SoapUI

Other less prominent implementations that were also considered, but did not have a compatible license, did not have an explicitly (public) license, are no longer developed and lacking in features, did not support WSDL 2.0 or are a combination of aforementioned, are: ActiveSOAP[6] and [36].

Apache Axis and Axis2

Apache Axis is an implementation of the SOAP (Simple Object Access Protocol) submission to W3C[15]. Axis has originated from the now obsolete Apache SOAP project. Axis stands for Apache EXtensible Interaction System. The latest version of Axis is 1.4, published in April 2006. Concurrently, there is a Release Plan for version 1.5, which is a maintenance release, aiming to resolve about 30 outstanding issues. Axis is essentially a SOAP engine; a framework for constructing SOAP processors such as clients, servers, gateways, etc. The features of Axis relevant to the WSDL-2TestFrame Application are the extensive support for the Web Service Description Language (WSDL) and emitter tooling that generates Java Classes from WSDL and generates WSDL documents from Java Classes.
B. Background Information of the Considered Implementations

Only the WSDL support and tooling for Java generation features are really required for the WSDL2TestFrame Application.

Apache Axis2 is the successor of Axis[16]. The advantages of Axis2 over Axis are[15]:

- **Speed** - Axis2 is based on the StAX API[7], which gives greater speed than the SAX event based parsing used in Axis1.x.

- **Stability** - Axis2 has fixed phases as well as user-defined phases for extensions. This allows far more stability as well as flexibility than Axis1.x.

- **Transport framework** - Transports (i.e., senders and listeners for SOAP over various protocols such as HTTP, SMTP, etc.), have been abstracted away from the Axis2 engine. Having a transport-independent Axis engine allows far more flexibility in transport options.

- **WSDL 2.0 support** - Axis2 supports both WSDL versions 1.1 and 2.0, which are used by Axis2’s code generation tools to create web service skeletons and client stubs.

- **Component-oriented architecture** - Axis2 components consist of handlers and modules in .mar and .aar archives. These easily reusable components allow extended functionality such as pattern processing for your applications or distribution to partners. Axis2 emphasizes the ”Module” concept over the ”Handler” concept of Axis 1.x. Modules contain handlers that are ordered by phase rules. These are attached to specific service(s).

Coupled with the Transport framework of Axis2 is the independent (XML) data binding feature; data binding is separate from the actual transportation and different data binding frameworks can be used, adding to the flexibility of Axis2. The supported data binding frameworks are ADB, XMLBeans, JiBX and JAXB.

The java.net JAX-WS Reference Implementation

The java.net JAX-WS Reference Implementation (RI) offer a broad future set with as main feature of is its implementation of the JAX-WS standard. JAX-WS stands for Java API for XML Web Services. The abbreviations JAX-WS and JAX-WS RI, the Reference Implementation of JAX-WS, are used interchangeably. The JAX-WS standard basically defines Java Annotations which minimizes the development effort required to implement Web Services, however the JAX-WS itself does not offer ample advantages within the client-side of Web Services. The features of JAX-WS RI relevant to the WSDL2TestFrame Application are again the extensive support for the Web Service Description Language (WSDL) and emitter tooling that generates Java Classes from WSDL and generates WSDL documents from Java Classes. A limitation of JAX-WS RI is that it currently only supports the JAXB data binding framework.
The Apache CXF framework

CXF contains a broad feature set, similar to the JAX-WS RI and is also JAX-WS compliant[18]. CXF JAX-WS support includes some extensions to the standard that make it significantly easier to use, compared to the reference implementation[20]: It will automatically generate code for request and response bean classes, and does not require a WSDL for simple cases. It also includes a ‘simple front end’ which allows creation of clients and endpoints without annotations. CXF supports both contract first development with WSDL and code first development starting from Java. CXF currently supports the JAXB, Aegis and XMLBeans data binding frameworks and does not support version WSDL 2, only 1.x.

soapUI

More information concerning soapUI can be found in section 3.2.2 and 7.1.2.

WS-TAXI

More information concerning soapUI can be found in section 7.1.2.

Data Bindings

Some of the considered implementations require a data binding framework. The relevant set of data binding frameworks is as follows[17][21]:

- **Axis2 Data Binding (ADB)** - The ADB framework is probably the simplest method of generating an Axis2 client. In most cases, all of the pertinent classes are created as inner classes of a main stub class. The ADB framework is very easy to use, but it does have limitations. It is not meant to be a full schema binding application, and has difficulty with structures such as XML Schema element extensions and restrictions.

- **XMLBeans** - Unlike the ADB framework, the XMLBeans framework is a fully functional schema compiler, so it doesn’t carry the same limitations as ADB. It is, however, a bit more complicated to use than ADB. It generates a huge number of files, and the programming model, while being certainly usable, is not as straightforward as ADB.

- **JiBX** - The JiBX framework is a complete data binding framework that actually provides not only WSDL-to-Java conversion, as covered in this document, but also Java-to-XML conversion. In some ways, JiBX provides the best of both worlds. JiBX is extremely flexible, enabling you to choose the classes that represent your entities, but it can be complicated to set up. On the other hand, once it is set up, actually using the generated code is as easy as using ADB.
B. Background Information of the Considered Implementations

- **JAXB (RI)** - JAXB (Java Architecture for XML Binding), another data binding framework, additionally uses Java’s annotations for augmenting the generated classes with additional information that bridges the gap between what is described by an XML schema and the information available (via Java’s reflection mechanisms) from a set of Java class definitions. The abbreviations JAXB and JAXB RI, the Reference Implementation of JAXB, are used interchangeably.
Appendix C

WSDL2TestFrame User Guide

This appendix contains the contents included of the short User Guide of the WSDL-2TestFrame Application directed to the developers and end-users.

C.1 Overview

WSDL2TestFrame is an application which aids the process of testing Web Services by the TestFrame method. Specifically, the WSDL2TestFrame aims to automate the generation and execution of Test Cases as much as possible in order to minimize the required time, effort and costs of the testing process.

Of importance to the WSDL2TestFrame are the following concepts or entities: the Apache libraries Axis2/XmlBeans, Web Service Description Language (WSDL) files, (TestFrame) Test Clusters and the TestFrame Engine.

- WSDL Files
  WSDL files define particular Web Services as defined by the W3 Consortium. Examples of details within a WSDL file are the various Web Operations of a Web Service, the input and output parameters of a Web Operation, and the (implicit) XML Schema of various parameters. Web Services are published by WSDL files and WSDL files enable clients or consumers to discover, determine and consume Web Services. WSDL2TestFrame uses WSDL files to analyze the structure of the input and output parameters of Web Operations by making use of the Axis2/XmlBeans libraries.

- The Axis2/XmlBeans libraries
  The Axis2/XmlBeans libraries, property of the Apache Foundation, are both utilized by the WSDL2TestFrame. With these libraries it is possible to, based on a WSDL file, generate Java source code capable of consuming or invoking Web Operations of the Web Service (defined in the WSDL file). Within the generated source code, two Classes, a Stub Class and a Callback Handler Class, contain methods which map one-on-one to the Web Operations. The remainder of the Classes contribute to the structures of the various
input and output parameters of the Web Operations. The generated source code will be stored in a sub-directory named ‘output’. WSDL2TestFrame analyzes this generated source code to generate a default Test Cluster.

- **Test Clusters**
  The TestFrame method is based on Test Clusters. Test Clusters mainly define Test Conditions, Test Cases and Action Words. When testing Web Services, it is common for Action Words to represent tests of Web Operations. WSDL2-TestFrame, just like the TestFrame Engine, accepts Test Clusters in the form of tab-delimited text files. These Test Clusters should be in a particular format specific to WSDL2TestFrame. The default Test Cluster, generated by WSDL2TestFrame, demonstrates this specific format. WSDL2TestFrame can automatically perform a Test Run with these Test Clusters without any additional coding. During the Test Run, the TestFrame Engine is also utilized for proper handling of the test results.

- **The TestFrame Engine**
  The TestFrame Engine is utilized by WSDL2TestFrame to compare the pre-defined expected output parameters with the actual output parameters of a Web Operation invocation and to generate a report of these comparisons. This contributes to the standardized way of documenting and evaluating test results.

### C.2 Configuration

WSDL2TestFrame reads settings from the configuration file: ‘Settings.txt’. This file is searched for in: ‘ for the Windows platform or in: ‘ for the Linux platform. WSDL2-TestFrame recognizes lines from the configuration file which have the following formats:

1. `set_dir_wsdl <<wsdl directory path>>`

   where `<<wsdl directory path>>` is the local path to the directory containing WSDL files to be read by WSDL2TestFrame. These files will be listed by their filename under the tab called ‘WSDL Files’. Any following configuration line which has this format will be ignored.

2. `set_dir_cluster <<cluster directory path>>`

   where `<<cluster directory path>>` is the local path to the directory containing Test Cluster files to be read by WSDL2TestFrame. These files will be listed by their
Using WSDL2TestFrame

filename in the list called ‘Cluster Files’. Any following configuration line which has this format will be ignored.

```
register_service <<qualified service name>>
```

where `<<qualified service name>>` is the common substring of the full Java Class names of the generated Axis2/XmlBeans Stub and CallbackHandler Class. For example, `<<qualified service name>>` could be `com.domain.web.WebService` with corresponding Stub Class `com.domain.web.WebServiceStub` and CallbackHandler Class `com.domain.web.WebServiceCallbackHandler`. This settings enables WSDL2TestFrame to locate the generated Axis2/XmlBeans Classes and subsequently, registering this Web Service. Every `<<qualified service name>>` will be listed under the tab Stub Class Names. Multiple configuration lines with this format are possible to register multiple Web Services. WSDL2TestFrame will display the `<<qualified service name>>` afterwards when generating Axis2/XmlBeans Classes.

Note that the TestFrame Engine has a separate configuration file.

### C.3 Using WSDL2TestFrame

WSDL2TestFrame has as input WSDL files, Cluster files, the ‘Settings.txt’ configuration file and the TestFrame ‘engine.ini’ configuration file. WSDL2TestFrame has as output template Cluster files and generated Axis2/XmlBeans Java source files, compiled Axis2/XmlBeans Java Classes, and a TestFrame Engine Report file. The use of WSDL2TestFrame consists of two phases: 1) the Test Generation Phase and 2) the Test Execution Phase. In the Test Generation Phase, a WSDL file is required as input and outputs the template Cluster file and generated Axis2/XmlBeans Java source files, compiled Axis2/XmlBeans Java Classes.

#### C.3.1 Test Generation Phase

Performing the Test Generation Phase entails the following steps by the user:

- selecting the desired WSDL file from the list under the ‘WSDL files’ tab on the left
- selecting the desired Cluster file from the list under the ‘Cluster Files’ section on the right
- pressing the button ‘Generate Test Cluster’ on the top

When selecting a WSDL file, WSDL2TestFrame will automatically pre-select the Cluster file which matches the filename of the WSDL file (by replacing the ‘.wsdl’ with the ‘.txt’ extension). If this Cluster file does not exist, the entry ‘<autogenerated>’ in the ‘Cluster Files’ section will be pre-selected. This entry means
that the matching Cluster file will be created. Note that the selected Cluster file will be overwritten by WSDL2TestFrame without any notice. It is recommended to give user-made Test Clusters a different filename than the matching Cluster file. The output of the Test Generation Phase is a template Test Cluster, exhibiting the global structure of a Test Cluster file of the particular Web Service based on the WSDL file.

C.3.2 Test Execution Phase

Performing the Test Execution Phase entails the following steps by the user:

- selecting the qualified service name of the desired Web Service from the list under the ‘Stub Class Names’ tab on the left
- selecting the desired Cluster file from the list under the ‘Cluster Files’ section on the right
- pressing the button ‘Run Test Cluster’ on the top

C.4 Editing Test Clusters

The generated template Test Clusters are a bare example of a Test Cluster and should be modified/supplemented by the user to create a proper Test Cluster. WSDL2TestFrame has a specific format of Test Clusters. This format is more clear when viewing an actual Test Cluster input file template. What follows next are the strict specifications of the format of a Test Cluster.

C.4.1 General Format

For each Test Cluster (input file), WSDL2TestFrame handles only the first recognized Test Condition section found in the input file. A Test Condition section is recognized as such by a line in the input file starting with the (whitespace-delimited) token ‘testcondition’ (without any whitespace prefix). Subsequent Test Condition sections are ignored. Within the Test Condition section, WSDL2TestFrame handles only the first recognized Test Case section. A Test Case section is recognized as such by a line in the Test Condition section starting with the (whitespace-delimited) token ‘testcase’ (without any whitespace prefix). Subsequent Test Case sections are ignored. Within the Test Case section, WSDL2TestFrame handles (multiple) Action Word sections. An Action Word section is recognized as such by a line in the Test Case section starting with a (whitespace-delimited) token which is not the reserved tokens ‘testcondition’ or ‘testcase’ (without any whitespace prefix).
C.4.2 Action Word Sections

The name of an Action Word should correspond to a name of a Web Operation listed in the WSDL file of the to be tested Web Service. These Web Operations are determined when generating the Axis2/XmlBeans Classes. The Action Word has the same structure as a Microsoft Excel table converted to a tab-delimited textual file. Action Words contain a number of parameter names and corresponding parameter values. The parameter name/value pair are referred to as a Parameter Entry. An Entry has a key (the parameter name) and a value (the parameter value). Within an Action Word section, every line is either a key or a value and every key is followed by a value. The order of the Parameter Entries does not matter, as they are sorted internally by WSDL2TestFrame. There are two kinds of Parameter Entries: Input Parameter Entries and Output Parameter Entries. Input Parameter Entries are recognized as such by prefixing the key with ‘i:’ Output Parameter Entries are recognized as such by prefixing the key with ‘o:’. The key is based on the structure of the generated Axis2/XmlBeans Classes, which in turn are based on the XML Schemas of the inputs and outputs, implicitly defined in the WSDL file. As such the key corresponds to a particular path within a certain XML Schema. Key and path can be be used interchangeably. A path consists of one or more non-whitespace tokens delimited by ‘.’, where each token represents an XML Node or a list of XML elements. A Parameter Entry with an empty path or a path with a prefixed or a postfixed ‘.’ will give a fault message and will be ignored. Values are tab-delimited tokens and each value represent a String value of a particular XML simple type element or XML Unit (an XML String, XML Integer, XML Date, etc.) as by XMLBeans heuristics. WSDL2TestFrame does not validate these String values; XmlBeans passes String parse errors on to WSDL2TestFrame, if any.

C.4.3 Action Word Example

The format of the key or path of a Parameter Entry requires additional explanation. While the format of the path is based on the heuristics of the XmlBeans framework, it is more comprehensible to explain this to start with an example Action Word.

Consider a Web Service which enables management of orders of customers in some company. Consider a Web Operation of this Web Service called ‘findOrder’ which finds an order by ID. This Web Operation returns a customer, list of products, price and payment method, in which the customer contains an ID and a list of previous order IDs and the list of products contains elements of product and product contains the product ID, the product price and the amount. An example of an Action Word of this Web Service could be:

```plaintext
1 i:ID
2 findOrder 39016 &Cont
3 o:Customer.ID
4 &Cont 1285 &Cont
5 o:Customer.OrderHistory[#]
```
The following observations can be made:

- The value of the first Parameter Entry is prefixed by the Action Word name.
- There are one input Parameter and eleven output Parameters.
- Two &Cont values are present on every value line except the last which has only one.
- ‘Customer’, ‘OrderHistory’ and ‘Product’ occur in multiple paths.
- A list is denoted by an array-like postfix, i.e. ‘[0]’, ‘[1]’ or ‘[#]’.
- Referred elements in a list do not need to be in ascending/descending order.
- The list ‘orderHistory’ is the last element in its path.
- The list ‘product’ is not the last element in its path.

In this example, the tokens ‘Customer’, ‘OrderHistory’ and ‘Product’ are names of XML Nodes. ‘ID’, ‘Price’, ‘Amount’ and ‘PaymentMethod’ are names of XML Unit. ‘OrderHistory’ is a list of XML Unit. Any token in a path which is not the last token of the path should always be a reference to an XML Node. If this is not the case, WSDL2TestFrame will give a fault message and ignore the Parameter Entry. XML nodes form a XML tree as defined in the (implicit) XML Schema. The last token of a path should always refer to an XML simple type/Unit. If this is not the case, WSDL2TestFrame will give a fault message and ignore the Parameter Entry. XML Nodes
and XML Units are created on-the-fly; an XML node/Unit is created(instantiated) only if a valid path of Parameter Entry contains a token of this XML Node/Unit. XML Nodes/Units that are not referenced by any path are not created(instantiated).

In the case of lists, a particular listed XML Node/Unit is referred to by the name of the list of XML Nodes/Units plus a list index. As such, the particular listed XML Node/Unit has no name, i.e. is anonymous. WSDL2TestFrame allows the verification of the expected size/length of a list. The size/length of a list is denoted by ‘[#]’.

The names of XML Nodes, XML Node lists, XML Units and XML Unit lists all have a separate namespace. XML Nodes and XML Units can always be distinguished from each other because of the position of the token in the path. Lists and non-lists can always be distinguished from each other because of the array-like postfix. For example, it is possible for an XML Node to have references ‘Product’, ‘Product[0]’, ‘Product.ID’ and ‘Product[0].ID’ in its sub-path if the XML Schema is defined as such.

C.4.4 Generating Test Cluster Templates

WSDL2TestFrame generates Test Cluster templates from implicit XML Schemas defined in the WSDL file. The set of all possible XML documents based on a particular XML Schema are of a much larger magnitude compared to the XML Schema it was based upon, in fact, this set may be infinitely large. The generated Test Cluster templates has to be limited in an appropriate manner. As such, WSDL2TestFrame abides the following rules when analyzing the XML Schemas:

- limit sequential structures to a length of 1, i.e. contain only one element
- omit all cyclic references of XML Nodes, i.e. an XML Node can node have a reference to a parent XML node
- omit nested XML Nodes of a depth larger than 30