# *Semantics based Assessment of the Robustness Quality Attribute of Web Services*

**By**

**Marwan Fayez Almomani**

**Supervisor**

**Dr. Samer Hanna**

**This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science**

**Deanship of Academic Research and Graduate Studies**

**Philadelphia University**

**August 2014**

# Semantics based Assessment of the Robustness Quality Attribute of Web Services

**By**

**Marwan Fayez Almomani**

**Supervisor**
**Dr. Samer Hanna**

**This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer Science**

**Deanship of Academic Research and Graduate Studies**
**Philadelphia University**

**August 2014**

Successfully defended and approved on _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

| Examination Committee Signature | | Signature |
|---|---|---|
| Dr.<br>Academic Rank: | , Chairman. | _ _ _ _ _ _ _ _ _ _ _ _ _ |
| Dr.<br>Academic Rank: | , Member. | _ _ _ _ _ _ _ _ _ _ _ _ _ |
| Dr.<br>Academic Rank: | , Member. | _ _ _ _ _ _ _ _ _ _ _ _ _ |
| Dr.<br>Academic Rank: | , External Member. | _ _ _ _ _ _ _ _ _ _ _ _ _ |

# *DEDICATION*

*I fully dedicate this work to:*

*To my father " Fayez Almomani"*

*And my mother "Ibtisam Almomani"*

*And my great wife "Duaa Almomani"*

*And my two sons " Hamzah and Abdalrahman"*

*And my brothers and sisters "Amer, Omer, Bothina and Rowida"*

*For being the most important part of my dream*

*For the support, courage, and unconditional love.*

***Marwan F. Almomani***

# ACKNOWLEDGMENT

Above all, Thanks to Allah, before and after everything, for giving me the knowledge and ability to complete this work in this final form.
It would not have been possible to write this master thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

I would like to express my thanks and sincere gratitude for who has guided me through my study and my thesis work; my supervisor Dr. Samer Hanna, for giving the wisdom, strength, support and knowledge in exploring things.

I would like to express my gratitude, my appreciation and my respect to the best Dr. in Philadelphia university Prof. Saeed AL-Ghoul for his humanity, his humility and his unlimited support for graduate students.

Also I would like to express my sincere thanks to the staff of the college who provide a warm and lively environment to encourage and help graduate students in their graduate study especially Dr. mo'uad ala'dami and Dr. Nameer Al-Emam.

Lastly, I would express my great thanks to all my friends at work and  at Philadelphia University.

*Marwan F. Almomani*

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| Abbreviation | Full Name |
|---|---|
| ASP | Active Server Pages |
| DTD | Document Type Definition |
| HTTP | Hypertext Transfer Protocol |
| IPA | Interface Propagation Analysis |
| J2EE | Java 2 Platform, Enterprise Edition |
| RPC | Remote Procedural Call |
| SOA | Service-Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| UDDI | Universal Description, Discovery and Integration |
| W3C | World Wide Web Consortium |
| WSDL | Web Service Description Language |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |

# LIST OF FIGURES

# ABSTRACT

Web Services are important for integrating distributed heterogeneous applications. One of the problems which is facing Web Services is the difficulty for a Service requester to trust Services provided by a Web Service provider.

One ideal way to increase trust between Service requesters and providers is testing. Testing can provide away to automatically assess quality attribute of a Web Service such as robustness based on its service description only, which in-turn will give Service requesters a mechanism to measure the desired quality for a specific Web Service.

This thesis proposes an approach that can be used to assess the robustness quality attribute of Web Services.

The approach is based on analysing Web Services Description Language (WSDL) documents of Web Services then build test cases based on input messages semantics and the input parameters data types to assess the robustness quality attribute of Web Services.

This approach will give a better test data since that it can give a real representation for a specific problem domain and can effectively test Web Services quality attributes.

# CHAPTER ONE: INTRODUCTION

## 1.1 Introduction

Recent years have seen an emergence of Service-oriented Architecture (SOA), that enable access to data that was difficult to reach, those Services are commonly referred to as Web Services, as they are available and provided over the Web. Web Services are designed as a way to automate application-to-application interactions; these applications are exchanging data via messages (De Virgilio, 2010) to accomplish various operations over a network.

Web Services are relatively new technology and have an increasing use within and across enterprises since they provide a framework for facilitating application-to-application interactions. Web Services became the most promising solution for integrating software applications over heterogeneous networks since they exchang information in a simple, standardized manner, and regardless of the platform (for example: Microsoft .NET, Sun J2EE, run on Window or Linux).

However, with the huge advantages of Web Services come a few challenges which restrict somehow the increase adoption of Web Services by companies and individuals within their businesses and limit the growing popularity of Web Services.

One of these challenges is the assessment of Web Service quality attribute such as robustness, Web Services can be offered publically by unknown providers through internet, and Service requesters cannot trust others software especially when they have no reputation, and there’s no way to access the source code of their Web Services because Service providers only provide the Web Service Description Language or (WSDL) of their Web Services for Service requesters. For these reasons the trustworthiness problem will arise between the Service requesters and providers.

## 1.2 Web Services advantages and challenges

Web Services have many benefits which made them becoming a major part in the development of Service oriented application (Sun, Li, Zhang, & Yan, 2009). These benefits put Web Services in front of the line and the best choice when talking about application-to-application interaction.

Some of the prominent features of Web Services are (Hanna & Alawneh, 2010) and (Cavanaugh, 2006):

- Interoperability: since Web Services are based on open standard such as XML, this will increase and facilitate the commutations and interactions between applications regardless of their platforms or programming language.
- Usability: any client can use the same Service as long as it provide the operations he need without the need of creating a custom Service for each requirement.
- Reusability: parts of the Service can be simply reused (Albreshne, 2009), this will have the effect of reducing time and cost required to design new Web Services. (Bozkurt, Harman, & Hassoun, 2010)
- Application and data integration: any Web Service can easily communicate with any other Service in a heterogeneous environment without the need for special software or to know the format used to store data or what platform and programming languages used to develop a particular Web Service.

However, with all the advantages of Web Services, come some challenges including but not limited to the following (Hanna S. , 2008):

- The trustworthiness problem: as mentioned before the Service requester only need the WSDL description file to access the desired Web Service. This looks as an advantage but this result to cause a problem, which is the trustworthiness problem (Dragoni, 2009).

  The trustworthiness problem can be addressed by the fact that Service requester has no mechanism to trust and ensure others Web Services since he has no means to access the source code, and current methods and testing techniques are inappropriate to work with this type of technologies.

- The selection problem: among the sea of Services that may provide the same functionalities, Service requesters have no way to find the right Service, there is no norm a Service requester can follow to choose the best Web Service among number of Services that provide the same functionalities (Makhlughian, Hashemi, Rastegari, & Pejman, 2012. ) In addition, WSDL does not describe the non-functional quality attributes such as robustness, security, and performance.

- Vulnerability to invalid inputs by malicious Service requesters: since Web Services are publically available over the internet, this situation presents several vulnerabilities, threats, and security issues, such as unauthorized access to sensitive data, parameter manipulation (malicious input) (Rani, Rao, & Devi, 2010).

## 1.3 Research Problem

The current Web Services architectures are facing some problems that are limiting their use and growth such as robustness and security. WSDL description only provide information about the functionalities of Web Services (the operations or the functions that a Web Service provides and how to bind to this Service), WSDL does not describe the non-functional aspects related to the quality of Service, and previous researches have shown that Web services are published to the public over the internet although they suffer from robustness issues (Laranjeiro, Vieira, & Madeira, 2014).So how can we test the non-functional requirement such as robustness of Web Services using WSDL description only?

Additionally, when building Web Services using different programming languages, the WSDL description will be different even for the same Web Services. For example the WSDL description for a Web Service implemented using Java will be different if we implemented it using ASP.net, so how can we distinguish and realize the WSDL specification generated by different programming languages and then generate test data accordingly?

And finally, when generating test data using the existing techniques, the generated test data may be correctly formed, but in most cases it doesn’t take into account the semantics that make test data more realistic, so how can we generate test data in such a way that satisfies both structural and semantics validity?.

The major question for this thesis is that: Can we automatically generate test data to assess the robustness quality attribute of a Web Service based on its contract regardless of what programming language was used to develop that Service?

## 1.4 Motivation

The work introduced in this thesis is stimulated by the following motivations:

- Tackling the above problems will allow service requester reaching high quality in the generated testing data.
- Providing an automatic way to assess the robustness quality attribute of Web services based on any WSDL.
- Allowing the service requester to evaluate how good a web service is.
- Providing a mechanism to select the most appropriate web service that best meet the requester needs among several web services that do the same task.
- Increasing the level of trust between service providers and service requesters through using testing.

## 1.5 Research Objectives

This thesis aims to generate test data in order to assess the robustness quality attribute of Web Services based on its interface or contract only and regardless of what platform was used to develop that Service.

## 1.6 Research Contributions

1. Developing an approach to assess the robustness quality attributes of a Web Service based on the semantics that can be extracted from WSDLs input messages, and the specifications of the input parameters data types inside the WSDL document of the Web Service under test.

2. Generating test data in more realistic way to assess the robustness quality of Web Services, based on any WSDL.

## 1.7 General structural Design

The approach can extract the WSDL document which attached with the Web Service itself, and then extract the text associated with each input message and the input parameters Data type. The approach will be able to generate test data automatically regardless of what platform a specific WSDL description belongs to, the generated test data will be saved in a XML document to be used in the testing process later.

## 1.8 Thesis Outline

The rest of this thesis consists of 5 chapters, organized as follows: Chapter two give the definitions for the main concepts related to this thesis and discusses the main components of Web Service and explains them in detail (such as XML-Schema ,WSDL ,UDDI ,SOAP). Chapter 3 give a detailed discussion on testing, testing techniques and Web services testing. Chapter 4 reviews general approaches used to generate test data in order to test Web services and assess the robustness quality attribute of Web Services. Chapter 5 present in details the proposed approach to assess the robustness quality attribute of Web Services. Chapter 6 summarizes the main achievements of this thesis, presents the general conclusions and suggests further research directions.

## 1.9 Summary

In this chapter, we have introduced a brief introduction about Web Services. We also discussed the importance of the Web Service for integrating distributed heterogeneous applications and describing briefly Web Services advantages and challenges, and finally introduced the motivations and objectives of this research and the contributions of the this thesis.

# CHAPTER TWO: BACKGROUND

## 2.1 Introduction

This chapter will give a full explanation of all Web Service components and also a detailed review of all concepts that are related to this thesis such as XML, XML Schema, SOAP and WSDL.

## 2.2 Web Services

Many studies gave different definitions of Web Services, There is no universally accepted definition of Web Services, as it has been under debate for quite some Time, For example (Bartolini, Bertolino, Marchetti, & Polini, 2009) defined the Web Service as an application that is published and accessed its functionality through Web.

Additionally (Hanna & Abu Ali, 2011) define Web Services as a new paradigm for building distributed software applications, which allow applications to interact, communicate, and exchange data regardless of what platform were used to implement it and also regardless of what programming language in which they were written.

Web Services are based on open standards such as eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL). It uses XML format which allow Services to communicate no matter what platform is used, and provide an effective way to reuse functionality, which in-turn reduces development time and cost (Casado, Tuya, & Younas, 2012).

## 2.3 The Web Services Model

There are three major roles within the Web Service architecture: Service provider, Service registry and Service requester. The interactions between these roles include three operations publish, find and bind operations as shown in figure 2.1 .

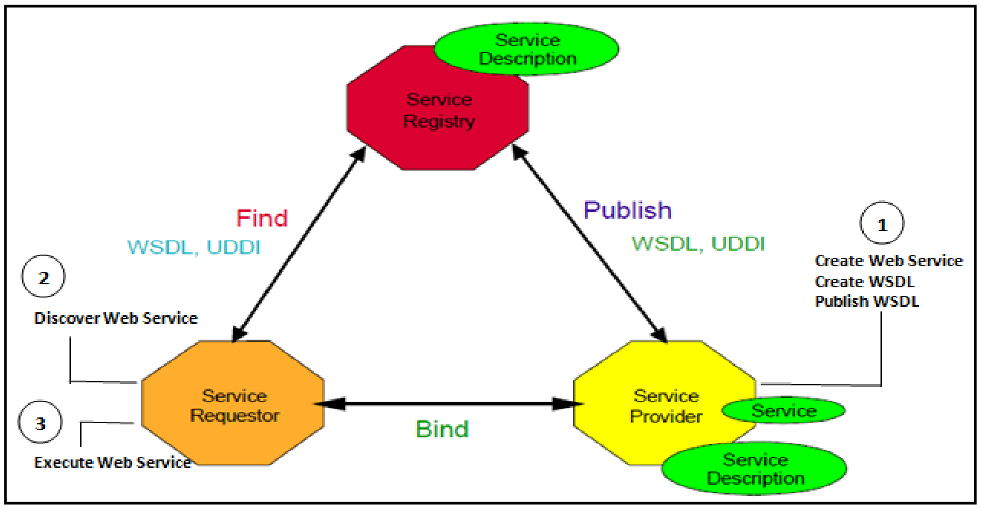

**Figure 2-1 illustrates Web Services roles together with the three operations.**

## 2.4 Web Service standards

This section will present a detailed explanation of all Web Services standards that are related to this thesis, XML, XML Schema, WSDL, SOAP, and UUDI.

### 2.4.1 XML:

XML is a document standard designed for storing, carrying, and exchanging information among incompatible applications (heterogeneous systems) because most of computer systems are originally programmed to understand or interpret such standards (Lee & Hwang, 2009).

Since the purpose of Web Services is for exchanging data between heterogeneous systems, all Web Services components are based on XML (i.e. WSDL, SOAP, UDDI), as a consequence for this, Web Services will have the ability to communicate with each other without the need for any special software. So XML is the backbone of Web Services.

### 2.4.2 XML schema:

XML schema describe the structure of an XML document , it's an alternative to Document Type Definition (DTD) but richer, more expressive and much more powerful.

According to (W3C, 2008), XML-Schema data type can be categorized into two types:

1. Simple data types:

Simple data types are including (Hanna S. , 2008) :

a. Built-in primitive data type (string, decimal, date, Boolean, time …).

b. Derived from built-in primitive data type: Derived from built-in primitive data types are actually built-in primitive data types with the addition of some default constraints to restrict the value space for a specific data type (De Melo & Silveira, 2011).

c. User-derived data types: User-derived data types are built-in primitive data types or derived from built-in primitive data types, with the addition of constraining facets were added to restrict the value space for a specific data type.

Table 2.1 gives a definition for all constraining facets and Table 2.2 shows the allowed constraining facets for each XML built-in data type.

**Table 2-1 Definition of constraining facets**

**Table 2-2 Summary of constraining facets for XML built-in data types**

| data type | Constraining facets | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | length | minL | maxL | wSp | Enum | MaxI | MaxE | MinI | MinE | pattern | Tdigits | Fdigit |
| String | • | • | • | • | • | | | | | • | | |
| ENTITIES | • | • | • | • | • | | | | | | | |
| ENTITY | • | • | • | • | • | | | | | • | | |
| ID | • | • | • | • | • | | | | | • | | |
| IDREF | • | • | • | • | • | | | | | • | | |
| IDREFS | • | • | • | • | • | | | | | | | |
| Language | • | • | • | • | • | | | | | • | | |
| Name | • | • | • | • | • | | | | | • | | |
| NCName | • | • | • | • | • | | | | | • | | |
| NMTOKEN | • | • | • | • | • | | | | | • | | |
| NMTOKENS | • | • | • | • | • | | | | | | | |
| NormalizedString | • | • | • | • | • | | | | | • | | |
| Notation | • | • | • | • | • | | | | | • | | |
| Token | • | • | • | • | • | | | | | • | | |
| AnyURI | • | • | • | • | • | | | | | • | | |
| QName | • | • | • | • | • | | | | | • | | |
| Base64Binary | • | • | • | • | • | | | | | • | | |
| HexBinary | • | • | • | • | • | | | | | • | | |
| Byte | | | | • | • | • | • | • | • | • | • | • |
| Decimal | | | | • | • | • | • | • | • | • | • | • |
| Int | | | | • | • | • | • | • | • | • | • | • |
| Integer | | | | • | • | • | • | • | • | • | • | • |
| Long | | | | • | • | • | • | • | • | • | • | • |
| NegativeInteger | | | | • | • | • | • | • | • | • | • | • |
| nonNegativeInteger | | | | • | • | • | • | • | • | • | • | • |
| nonPositiveInteger | | | | • | • | • | • | • | • | • | • | • |
| PositiveInteger | | | | • | • | • | • | • | • | • | • | • |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Short | | | | • | • | • | • | • | • | • | • | • |
| nsignedLong | | | | • | • | • | • | • | • | • | • | • |
| unsignedInt | | | | • | • | • | • | • | • | • | • | • |
| unsignedShort | | | | • | • | • | • | • | • | • | • | • |
| unsignedByte | | | | • | • | • | • | • | • | • | • | • |
| Double | | | | • | • | • | • | • | • | • | • | • |
| Float | | | | • | • | • | • | • | • | • | • | • |
| Boolean | | | | • | | | | | | • | | |
| Date | | | | • | • | • | • | • | • | • | • | • |
| DateTime | | | | • | • | • | • | • | • | • | • | • |
| Duration | | | | • | • | • | • | • | • | • | • | • |
| gDay | | | | • | • | • | • | • | • | • | • | • |
| gMonth | | | | • | • | • | • | • | • | • | • | • |
| gMonthDay | | | | • | • | • | • | • | • | • | • | • |
| gYear | | | | • | • | • | • | • | • | • | • | • |
| gYearMonth | | | | • | • | • | • | • | • | • | • | • |
| Time | | | | • | • | • | • | • | • | • | • | • |

d. List data types: allows the creation of a type that has a value set that consists of a finite length list of acceptable values of the same type, rather than just one.

e. Union data types: joining two simple types or more together to create a new simple type.

2. Complex Data types: consist of one or more elements and attributes of data types (Hanna & Alawneh, 2010) and (De Melo & Silveira, 2011). Just like simple data types complex data types can have restrictions also using sequence, choice, and all.

### 2.4.3 Web Service Description Language (WSDL):

WSDL is a standard document written in XML, which provide a way for describing Web Services. It provides a description that specifies the location of the Services and their capabilities (Noikajana & Suwannasart, 2009).

A WSDL file contains all of the information necessary for a client to invoke the operations of a Web Service including:

- Interface information: what a Web Service can do.
- The input and output messages used by an operation.
- Binding information: describing the transporting protocols used to communicate, and messages format allowed for each operation.
- Address information: where to locate the requested Web Service.
- Data types information for all requests and responses messages.

A WSDL document contains the following elements (W3C, 2010):

1. Definitions element:

   The *definitions* element is the root element, it defines the Web Service and act like a container of all other WSDL elements.

2. Types element:

The *types* element describes all the data types of the parameters of the input, output and exceptions messages.

3. Message element:

   This element is used to describe the structure of all messages types; input, output and exceptions messages, exchanged between the Service requester and Service provider including input parameters and their data types.

4. Porttype element:

   The *porttype* element is a container of all the operations provided by a Web Service and what messages are involved within these operations.

5. Binding elements:

   This element describes how the operations provided by a Web Service transmitted over the network, and it also defines the message type RPC or document.

6. Service element:

   This element tells the client where a Web Service can be accessed using which port.

### 2.4.4 Simple Object Access Protocol (SOAP):

Is an XML-based standard messaging protocol for exchanging structured information using HTTP as main means transportation in a decentralized, distributed environments. SOAP defines the three types of messages supported by WSDL (input, output, and exceptions messages) in XML by which the messages are guaranteed to be compatible and understood by any application (Cavanaugh, 2006).

SOAP message has an envelope element which is the root element, and it has two child elements; an optional header and a mandatory body.

### 2.4.5 Universal Description, Discovery and Integration (UDDI):

Universal Description, Discovery and Integration or UDDI for abbreviation, is a distributed global registry where Service providers can register their Web Services using WSDL, so that clients can find them (Kumar & Varalakshmi, 2012).

## 2.5 Web Services robustness quality attribute

The software quality attribute that is of interest to this thesis is robustness which is a sub attribute of reliability (Hanna S. , 2008).

Reliability is defined as:

> *"The quality of a system to provide consistent results, effectively handle variation in inputs and environmental conditions, and recovery gracefully from error conditions"* (O'Brien, Merson, & & Bass, 2005).

The previous definition of reliability shows that reliability is mainly related to robustness and fault-tolerance.

It's worth mentioning here that to achieve robustness, some testing technique such as robustness testing are required (testing techniques will be discussed in depth next section).

Robustness is defined as:

> *"The degree to which a system or component can function correctly in the presence of invalid input or stressful environmental conditions"* (IEEE, 1990).

(O'Brien, Merson, & & Bass, 2005) and (Hanna S. , 2008) mentioned that the level of trust between Service providers and Service requesters, the ability to distinguishing and ranking Services with similar functionalities can be increased by assessing the robustness quality attribute of Web Services.

## 2.6 Testing

For any software, it is essential to evaluate its functionalities, detect any possible errors, and assess it quality before using it.

Testing is one way that enables us to know if a program reacts the way we assume and produce the expected output for a given set of inputs.

Testing can be defined as:

> *"Software testing is a quality assurance process that is part of the verification and validation processes, and involves executing the system under test with test data for the purpose of detecting faults and assessing the quality attributes of that system or software component"* (Hanna S. , 2008).

Testing can be functional or non-functional testing, where functional testing verify and validate a particular functionality of the code, and non-functional testing assess the non-functional requirements of a software such as robustness, performance, and security.

As our approach of testing Web Services aims to assess the robustness quality attribute of Web services, the type of testing used in this thesis is classified under non-functional testing.

## 2.7 Testing techniques:

As mentioned before, software testing is an important way for assessing the software in order to determine its quality.

Many testing techniques have been proposed and categorized into various types each following specific criteria, such as the availability of source code, the intent of the test, the level of testing, and the quality attribute of system behavior (Hanna S. , 2008).

The testing techniques that most related to this research are:

- Boundary value testing: is a black-box testing techniques because accessing the source code is not permitted, and the test data is generated in the absence of the source code. On the contrary, when the presence of source code is mandatory to perform testing, this type of testing is considered to be a white-box texting (Khan, 2011) and (Khan, 2012) .

Boundary value testing is used to identify errors that may occur at the boundaries of an input domain where test cases are selected at the edges of a specific test class.

- Equivalence class partitioning: is also a black-box testing techniques, and mainly used to reduce the number of the generated test cases by dividing the input domain data into a finite set of different equivalence data classes (Huang & Peleska, 2013).

- Interface Propagation Analysis (IPA): IPA is a fault injection based technique, used to assess robustness, reliability, and Security quality attributes. IPA injects corrupted or perturbed information into the data, and then IPA analyze the behavior of the system by observing how that corrupted information propagates through the system (Jorgensen, 2013).

- Boundary Value Based Robustness Testing: is an extension to boundary value testing, generating test cases are based on choosing values around the boundaries of the input parameter i.e. a value above the maximum and below the minimum value of a specific input parameter (ANUPRIYA, SHARMA, SEEMA, & DEEPTI, 2010).

- Syntax testing: is a black box testing technique, syntax testing helps the tester to be sure that input values are being checked correctly using some formal description methods such as regular expression (Moreira, Antunes, & Ramal, 2010).

.

Table 2.3. shows how test data are generated using the pre-mentioned testing techniques.

**Table 2-3 Test data generation methods**

| **Testing technique** | **description** |
|---|---|
| IPA | Injects corrupted or perturbed information into the data during the execution of a set of valid input values. |
| Boundary Value Based Robustness testing | choosing values around the boundaries of the input parameter i.e. a value above the maximum and below the minimum value of a specific input parameter |
| Syntax testing | Breach the rules of formally-defined syntax of the input parameters |
| Equivalence Partitioning | Partitioning the input domain into sets or classes of input data then generate test data according to these classes. |

## 2.8 Assessing Web Services quality attribute

Like all software, Web Services need to be tested; they need to be very robust and reliable. Testing provide a way to test and assess the quality attributes of Web Services.

However, testing Web Services is more difficult than testing traditional software. Web Services testing faces many challenges and difficulties, these challenges and difficulties came from:

1. The complex nature of Web Services, due to the fact that Web Services are distributed applications based on different protocols such as UDDI and SOAP, and depending on limited specifications provided by WSDL (Bozkurt, Harman, & Hassoun, 2010) and (De Melo & Silveira, 2011).

2. The absence of source code, enforcing test to be based on the published descriptions by which the test process could be very costly and error-prone (De Melo & Silveira, 2011).
3. Web Services do not display a user interface that can be tested.
4. WSDL does not describe the non-functional aspects related to the quality of Service.

To overcome these challenges, testing is used by adopting existing traditional testing techniques and employs them to test Web Services (De Melo & Silveira, 2011).

## 2.9 Literature Review for Web services Testing

### 2.9.1 A Technique for Deploying Robust Web Services

(Laranjeiro, Vieira, & Madeira, 2014) Presented an approach to automatically detect robustness problems and fix these issues.

The proposed approach consists of many phases which can be summarized in the following steps:

Phase 1: in this phase the WSDL file is first examined in order to extract information related to the Web Service under test, these information are all the operations provided by the Web Service, input messages parameters, and their data types, then determining the valid domains and any constraining facets for each input parameter through searching the associated XSD and using an extension to XML schema which express the valid domains for each input parameter and the meaning for domain dependencies between parameters.

Phase 2: in this phase a synthetic workload is generated in order to test each operation where a set of valid inputs are generated randomly for every input parameter for each operation.

Phase 3: in this phase the proposed approach identify robustness problems for the Web service under test by using a fault injection process during the execution of the workload generated in the previous phase. The faults generated using a mutation rules based on input parameters data types.

Phase 4: this phase consists of fixing any robustness problems revealed in phase 3 through the use of proposed protective scheme.

### 2.9.2 An Approach for WSDL-Based Automated Robustness Testing of Web Services

(Hanna & Munro, 2009) Proposed an approach to generate test data by analyzing WSDL document and extracting useful information that can be used to generate test data.

The test data generation strategy is based on the input parameters data types and their constrains. Input parameters data types can be classified in to primitive simple data types (string, numeric, data-time, and Boolean), user-derived data types, and complex data type.

(Hanna & Munro, 2009) Technique of generating test data can be summarized as follows:

1. Generating test data for primitive simple data types: valid and invalid test data are generated by changing the data type of the input parameter, inserting null or empty parameter, or using the boundaries limits of values by following a pre-defined test case generation rules. For example: for the input parameter of type numeric, date-time, and Boolean, invalid test cases are generated by replacing or changing the data type of the input parameter to string.

2. Generating test data for user-derived data types: test cases generated for user-derived data types depends on the base type of the input parameter data type and the constraining facets. . For example: for the input parameter of type decimal or float, valid and invalid test cases are generated by using numeric boundaries constraining facets such as minInclusive, minExclusive, maxInclusive, and maxExclusive.

3. Generating test data for complex data types: test cases generated for complex data types by decomposing complex data types in to simple and user-derived data types then generating test data according to what mentioned in step 1 and 2. The resulting test cases are computed.

### 2.9.3 Automatic Web Service robustness testing from WSDL descriptions

(Salva & Rabhi, 2009) Introduced an approach to test Web Services robustness using the information provided by WSDL only, it based on analyzing the Web Service observability to determining what type of hazards (unspecified events) that can be used for testing, and which of them are blocked by the SOAP processor.

The presented approach assumed that a specific Web Service can be considered robust if and only if all its operations are robust too, the robustness of an operation can be determined according to its behavior when invoking it with hazards.

(Salva & Rabhi, 2009) Classified the type of hazards that can be used for testing, for example replacing, adding, deleting, or inverting the parameter type are classified as blocked by SAOP hazards and cannot be used for testing, in the other hand, unusual values (such as special characters) passes the SAOP processor, therefore, it can be used for testing.

The proposed approach can be summarized as follows:

1. Generating test cases to check if all operations specified in the WSDL description file are exists by and operates as expected, making sure that each operation can be called using a value representing the parameter type expressed in the WSDL file, and returning the expected response. Following this step the authors ensure the correctness of the functionality for all operations.

2. With the presence of hazard (unusual values), test cases are generated to analyze the Web Service behavior to see its reaction and to check if each operation does not crash or hang by calling it with unusual values and to make sure it response with a proper SOAP fault.

## 2.10 Summary

In this chapter, we gave a definition of Web Service and introduced a background about Web Services components and its roles; we also discussed XML and XML-Schema. We also gave a detailed discussion for the Web services standards, and a focus discussion on

WSDL which is the most important standard to the Web Service testing approach that is developed in this thesis.

Also gave a detailed discussion on quality attributes and software testing techniques. A definition of reliability and their related sub attribute was given in this chapter.

And finally we discussed some recent studies and researches aiming to introduce different approaches and techniques for testing Web Services.

# CHAPTER THREE: A WSDL-BASED APPROACH TO ASSESS THE ROBUSTNESS OF WEB SERVICES

## 3.1 Introduction

This chapter describes our approach of assessing the robustness quality attribute of Web services. The approach depends on analyzing WSDLs documents to find the name of the different input messages and after that generating test data based on analyzing the text associated with each input message. Our approach will consider both the syntax and semantics in the process of generating test data.

## 3.2 Analyzing WSDL

As mentioned before, WSDL consists of several elements; the elements that are of interest to our approach are PORTTYPE, and TYPES elements

The PORTTYPE element contains information related to the operations provided by a Web Service and what input messages are involved within these operations. Each input message has an associated text which determine and tells what specific inputs is required for this input message. In the other hand, The *TYPES* element describes the input parameters data types.

Our approach starts by analyzing the WSDL file to find specific useful information; it first extracts every input message name which can be found in the PORTTYPE element and analyzes the text associated with it, looking for descriptive text that can be used to determine the different types of the input messages and use it in the process of generating valid and invalid test cases. In addition, our approach will extract the data types for the input parameters and generate valid and invalid test data accordingly with the use of their associated constraining facets.

The proposed approach is mainly focuses on using the semantics of the input messages to generate test data and then uses these test data to assess the robustness of a specific Web Service by analyzing its behavior in the presence of invalid inputs.

Figure (5.1) shows how WSDL analyzing process works.

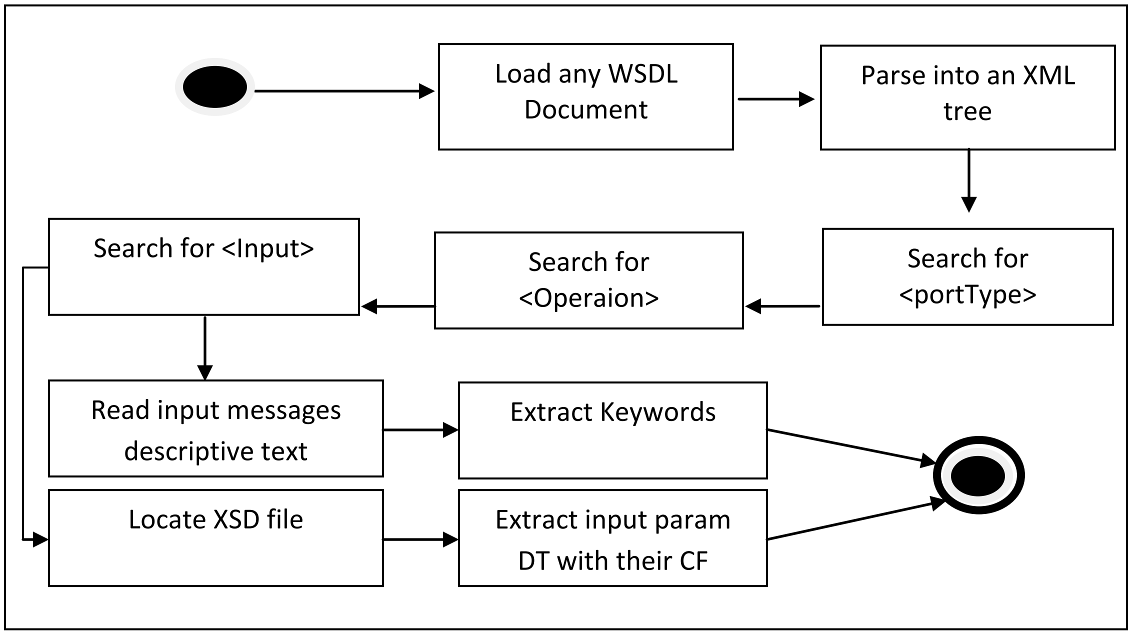

**Figure 3-1 WSDL analysis workflow**

## 3.3 Analyzing Text associated with Input Messages

Each input message has a descriptive text; our approach will read that text in order to locate keywords, the extracted keywords will be used to generate valid and invalid test data.

A descriptive text can be classified into three categories:

1. Keywords that can be mapped to a pre-defined regular expression; in this case the descriptive text associated with the input message contains a keyword, this keyword is previously mapped to a valid regular expression which express the meaning (semantic) of this keyword.

2. Keywords with a pre-defined value range; in some cases the keyword semantic cannot be expressed using a regular expression such as the keywords year and weather. In such cases our approach will depend on a pre-defined value range which represents the accepted values for such keywords. This values range will be used to generate valid and invalid test data based on selecting values within and outside the range of the value range.

3. No keyword can be found; in this case the descriptive text associated with the input message has no word that can be expressed using regular expressions and a value range cannot be defined. To handle this situation our approach will depend only on the input parameter data types, constraining facets, and testing techniques to generate test data.

## 3.4 Building Keywords dictionary

To build a keyword dictionary, manual examination need to be done on every available WSDL file belonging to a specific Web services domain.

As an example, we perform the examination process on a real sample for more than 250 WSDLs. After an extensive study for the sample we successfully found that WSDL can provide us with very helpful keywords that can be used to generate valid and invalid test data.

Our survey successfully found keywords that can be assigned to a regular expression or value ranges which in-turn used to generate valid and invalid test data.

We also were able to build regular expressions and define a value range for the extracted keywords. Table 5.1 shows  the extracted keywords after analyzing 250 WSDLs which can be used to generate test data by assigning them to regular expressions or range of values.

**Table 3-1The extracted keywords**

| Keywords | |
|---|---|
| Bank | mobile numbers |
| Movie | Time |
| Cities | (ISBN) |
| Country | Blood Group |
| NameState | URL |
| Vehicle | Username |
| Quote | Password |
| DollarValue | Email Address |
| Temperature | HTML |
| Humedad | Currency |
| Version | Timestamp |
| Count | User ID |
| Size | IBAN |

| | |
|---|---|
| Continent | Date and Time |
| Language | SSN |
| Taxrate | ip addresses |
| Year | Credit card / Debit card |
| SSN | Mastercard |
| Number | Discover Card |
| Exchange rate | Phone Numbers |
| Bandwidth | international phone number |
| XML | Time |
| Birthday | Email |
| Price | Credit Card |
| Payment | Time |
| Balance | Month |
| Studentbirthdate | Phone Numbers |
| Employbirthdate | Date |
| ZipCode/Zip | Timezone |
| Currency | Bank Swift Number/BSN |
| Decimals | IP Address |
| PO Box | Website |

The extracted keywords table 5.1 is used as a dictionary, this dictionary can be used to know if any text associated with input messages has a keyword or not, then Using the steps in the developed algorithm presented in section 5.7 we can generate valid and invalid test data by depending on the semantics extracted from the text associated with input messages and input parameters data types with their constraining facets. It's worth mentioning here that the size of the dictionary can be infinite, so for each specific Web services domain a different dictionary will be constructed in order to reduce the generated dictionary size.

## 3.5 A Model for Robustness Testing of Web Services

This section will give a detailed explanation for the model of our approach for robustness assessment of Web Services and how it works (see Figure 5.2).

The goal of our approach is to generate valid and invalid test data to assess the robustness quality of Web Services from the client side, and then analyzing the behavior of a Web Service under test in presence of invalid inputs.

Figure 5.2 represents the model of our approach for generating test data. First, the WSDL file is analyzed to extract useful information which can be used in the process of generating test data. The specific information inside WSDL that our approach will look for is: input messages name and the text associated with each input message, and input parameter data types with their constraining facets.

The first information can be located in the PORTTYPE element of WSDL. As mentioned before, the PORTTYPE element is a container of all the operations provided by a Web Service, and it contains the input, output, and exception messages involved within these operations.

What is important here is the input messages, our approach will extract the input message name together with the text associated with each input message. Text associated with each input message is used to determine the semantic (type) of what expected as inputs for this input message.

The second information can be located in the TYPES element of WSDL, which holds a description of the input parameters data types.

In order to generate test data, our approach will analyze the extracted text associated with each input message and classifying it into descriptive and non descriptive text.

A descriptive text contains keywords such as email, phone, SSN … etc, each keyword will be mapped to an appropriate pre-defined regular expression or pre-defined value range, then the regular expression or value range will be used to generate valid and invalid test data. Test data generation method based on regular expressions and value ranges will be discussed in depth later.

For non descriptive text, no action will be taken since our method depends mainly on finding keywords and generating test data for such kind of text is impossible. To handle this situation our approach will depend on input parameter data types and their constraining facets to generate test data

For each input parameter a test data is generated based on its data type expressed in the TYPES element of WSDL and the defined constraining facets for each data type, the test data generated using this method will follow pre-defined rules based on testing techniques. These rules and the test generation method based on input parameter data types will be also discussed in depth later.

Finally the generated test data will be exported into XML document. This XML document will be used to test the robustness of Web Services. The robustness quality will be measured according to the behavior of a Web Service in the presence of invalid inputs.

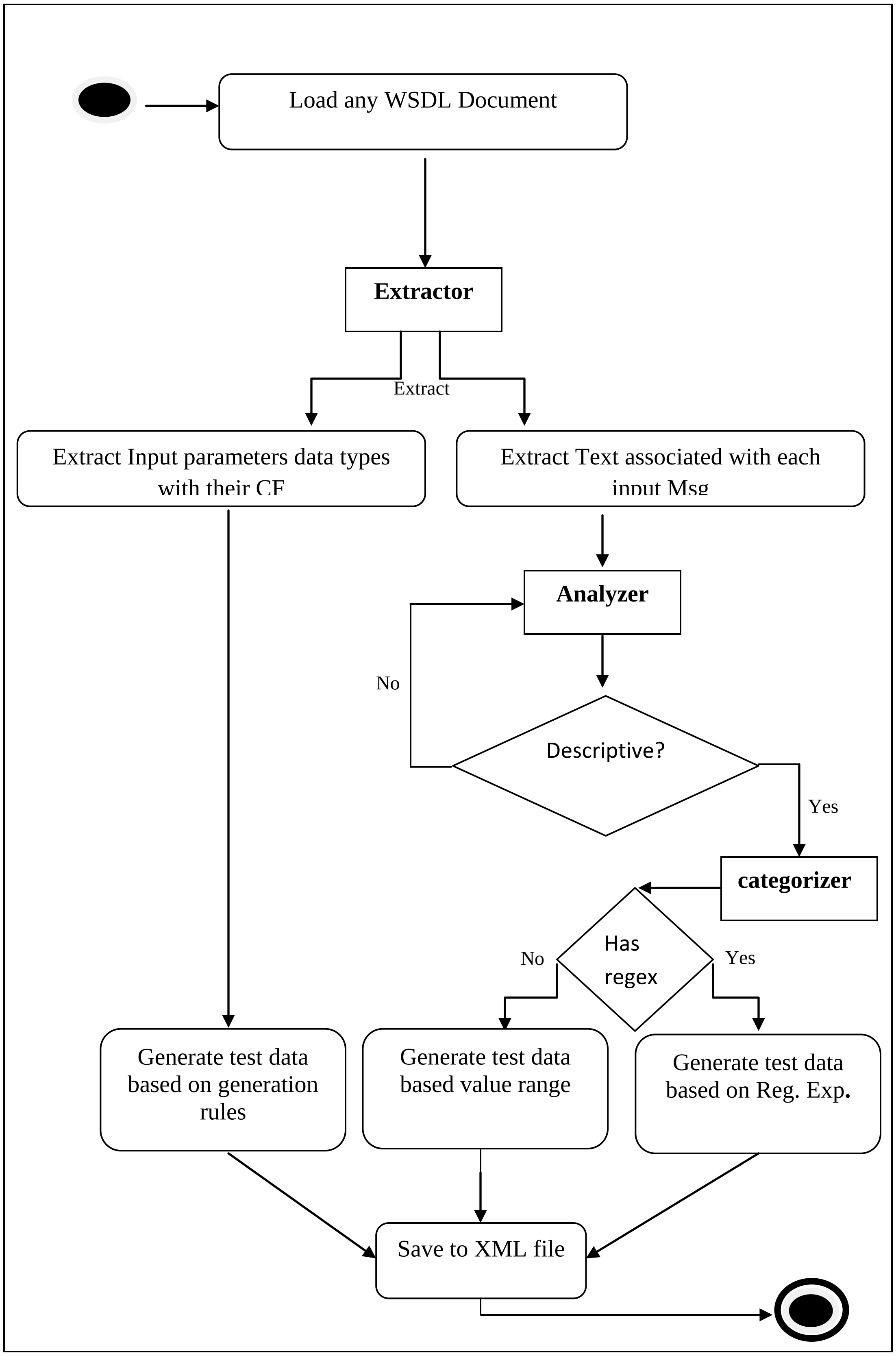
Load any WSDL Document
Extractor
Extract
Extract Input parameters data types with their CF
Extract Text associated with each input Msg
Analyzer
No
Descriptive?
Yes
categorizer
Has regex
No
Yes
Generate test data based on generation rules
Generate test data based value range
Generate test data based on Reg. Exp.
Save to XML file

**Figure 3-2 The model**

Our model consists of many components, these components are explained next in more details:

**WSDL document**: is the description file of the Web Service under test.

**Extractor**: responsible for performing the WSDL analysis process presented in figure 5.1. This component extract text associated with each input message, and each input parameter data type with their constraining facets.

**Analyzer**: extract words from the extracted text, and then examine these words to make a decision if those words are keywords or not (see section 5.3).

**Categorizer**: determine the type of the extracted keywords in order to map those words to regular expression or to a pre-defined value range.

**Generate test data based on regular expression**: responsible for generating valid and invalid test data based on pre-defined regular expression (see section 5.4).

**Generate test data based on pre-defined value range**: responsible for generating valid and invalid test data based on pre-defined value range (see section 5.5).

**Generate test data based on generation rules**: responsible for generating valid and invalid test data based on pre-defined rules which generate test data according to the input parameter data type and its constraining facets (see section 5.6).

**Test data**: the generated test data using the previous methods are saved in an XML file, this file is used to test Web Service under test by sending the generated test data in a SOAP messages.

## 3.6 Generating Test Cases for text associated with input messages based on regular expression

As mentioned before, text associated with input messages can be meaningful, we can generate test data based on it.

Our approach starts by analyzing WSDL file until it find input messages, each input message holds a text, which describe the expected types of the inputs for a specific input message. For each input message, the text associated with it will be analyzed in order to find keywords (email, SSN, phone … etc. ), the extracted keywords may help in determining the type of inputs expected for this input message.

When a keyword is found our approach will perform a mapping between the keyword and a pre-defined regular expression which express the meaning of this keyword.

Figure 5.3. illustrates an example of the PORTTYPE element of a WSDL file for a Web Service provide a registrations Service, as shown, the PORTTYPE element consists of portType name (“Registrations”), an operation (“RegistrationInfo”), and an input message.

According to our approach, a text can be considered as a keyword if it already defined in the dictionary which we build after a survey for more than 250 WSDL files (see table 6.1 ).

Each input message has a descriptive text; for example *<wsdl:input message="tns:Email"/>,* the word Email is a keyword since it can be expressed using a regular expression.

After extracting the associated text, the text is analyzed to find keyword. Our approach then relates each keyword which determines the input message expected input to a regular expression that defines the meaning of this keyword.

Our approach uses the regular expression to generate valid and invalid test data. The valid test data are simply generated by applying the regular expression definition for each keyword. In the other hand, to generate invalid test data a perturbation strategy is used by applying a modification rules on the regular expression of valid inputs itself.

Our perturbation strategy performs modification operations to modify the regular expression:

1. Disordering the sets in a regular expression.
2. Changing the appearance time of selected elements from a set in a regular expression.
3. Deleting sets from a regular expression.
4. Replacing mandatory sets with an optional sets in regular expression and vice versa.

5. Adding invalid characters such as “{“ string to the a regular expression.
6. Replacing the original regular expression with another.

By applying the previous perturbation rules on a regular expression we can get 6 different invalid test data.

The following example illustrate our approach for generating test cases for the text associated with input messages based on regular expression:

Suppose we have WSDL file for a specific Web Service that provide a registration Service and we want to assess if this Web Services is robust or not. Figure 5.3 shows the PORTTYPE element of the WSDL file for this Web Service.

```
<wsdl:portType name="Registration">
<wsdl:operation name="RegistrationInfo">
<wsdl:input message="tns:Email"/>
…
</wsdl:operation>
…
</wsdl:portType>
```

**Figure 3-3 Example of input messages with its descriptive text**

The part of the PORTTYPE element which is of special importance to our approach is the input message name and its associated text. Generating valid and invalid test cases for this example can be summarized in the following steps:

1. Locating input messages inside WSDL and Extracting the descriptive text for each input message. i.e. <wsdl:input message="tns:Email"/>
2. Analyzing the descriptive text i.e “Email”, in order to determine the input message type.
3. Locating keywords; the associated text in our example has only one word (Email), Email is a keyword since it has a pre-defined in the dictionary and it has a regular expression.

4. Each keyword is related to a proper regular expression which define the valid inputs for the input message. A valid regular expression for the keyword Email can be: [w- \ .]+@ ([\w-] + \.) + [\w-] {2,4}, so our approach will relate the keyword Email to this regular expression.

5. Using the previous Email regular expression, our approach will generate valid test data by simply apply the rules defined for that regular expression. For example:

   TestEmail@testcomp.com; is a valid test data generated from using email regular expression.

6. For the generation of invalid test data, we will use the pre-mentioned modification rules to modify the email regular expression. For example table 5.1 shows how we can get the invalid test data after applying the perturbation rules on the email regular expression:

**Table 3-2 Invalid test data generation using perturbed email regular exp.**

| Rule No. | Invalid email regular exp. | Invalid test data |
|---|---|---|
| 1 | @[w- \ .]+([\w-] + \.)+[\w-] {2,4} | @TestEmailtestcomp.com |
| 2 | [w-\.]+@([\w-] + \.) * [\w-] {2,4} | TestEmail@.com |
| 3 | [w-\.]+([\w-] + \.) +[\w-] {2,4} | TestEmailtestcomp.com |
| 4 | [w-\.]+@?([\w-] + \.) +[\w-] {2,4} | TestEmailtestcomp.com |
| 5 | [w-\.](\s)+@([\w-]+\.)+[\w-]{2,4} | TestEmail{@testcomp.com |
| 6 | ^(19\|20)\d\d[- /.](0[1-9]\|1[012])[- /.](0[1-9]\|[12][0-9]\|3[01])$ | 2011-0812-072531 |

## 3.7 Generating Test Cases for text associated with input messages based on pre-defined values domain

In section 5.3. we have mentioned that a descriptive text for an input message can be classified into three categories; keywords that can be assigned to a regular expression, keyword that can be assigned to a value range, and non descriptive text.

In this section we are going to give a detailed discussion for the second category “keyword that can be assigned to a value range”. Our approach will follow the same steps introduced in section 5.4. but instead of assigning keywords to a regular expression it will assign them to a pre-defined value range.

For each input message, the text associated with it will be analyzed in order to find keywords (year, weather, city… etc), the extracted keywords may help in determining the type of inputs expected for this input message.

When a keyword is found our approach will perform a mapping between the keyword and a pre-defined regular expression or to a value range which represents the accepted values for this keyword.

Figure 5.4. illustrates an example of the PORTTYPE element of a WSDL file for a Web Service provides a weather Service, as shown, the PORTTYPE element consists of portType name (“GlobalWeatherSoap”), an operation (“GetWeather”), and an input message (“GetWeatherHttpGetIn”). This Service is simply gives the expected temperature for a specific city.

```
<wsdl:portType name=" GlobalWeatherSoap ">
<wsdl:operation name=" GetWeather ">
<wsdl:input message="tns: GetWeatherHttpGetIn "/>
…
</wsdl:operation>
…
</wsdl:portType>
```

**Figure 3-4 Weather Web service PortType**

In the previous example the GetWeather operation has one input message which has a descriptive text; GetWeatherHttpGetIn.

The descriptive text is analyzed to find keywords, for example the word Weather can be extracted from the descriptive text which considered as a keyword since its already defined in our dictionary.

Following our approach of assigning each keyword to a proper regular expression will be useless in such cases because regular expression for such keywords cannot be used to generate valid and invalid test data. Instead, our approach uses a pre-defined value range which represents the expected values and its data type that can be concluded from the semantic of such keywords.

Since our approach aims to assess the robustness quality attribute of Web Services we will use the robustness testing technique (see section 3.5) to generate valid and invalid test data for such keywords. According to robustness testing, generating test data will follow the rules as shown in table 5.2:

**Table 3-3 Generating test data using robustness testing**

| Rule | Description |
|---|---|
| Min | Valid test data, exactly the minimal value for a specific domain |
| Min+ | Valid test data, just above the minimal value for a specific domain |
| Nom | Valid test data, average value for a specific domain |
| Max | Valid test data, exactly the maximum value for a specific domain |
| Max- | Valid test data, just below the maximum value for a specific domain |
| Min- | Invalid test data, just below the minimal value for a specific domain |
| Max+ | Invalid test data, just above the maximum value for a specific domain |

The previous rules will generate seven different test data, two of them are invalid test data and the rest are valid test data.

These test data will be used to test a Web Service and to assess the robustness quality attribute for the Web Service under test in the presence of invalid inputs.

## 3.8 Generating Test Cases for non-descriptive text associated with input messages

Considering the following PORTTYPE element of WSDL file for a simple Web Service that convert Fahrenheit to Celsius and vice versa, the PORTTYPE element consists of two operation, two input messages, and two output messages as shown in figure 5.5.

```
<wsdl:portType name="ConvertSoap">
<wsdl:operation name="FahrenheitToCelsius">
<wsdl:input message="tns:FahrenheitToCelsiusSoapIn"/>
<wsdl:output message="tns:FahrenheitToCelsiusSoapOut"/>
</wsdl:operation>
<wsdl:operation name="CelsiusToFahrenheit">
<wsdl:input message="tns:CelsiusToFahrenheitSoapIn"/>
<wsdl:output message="tns:CelsiusToFahrenheitSoapOut"/>
</wsdl:operation>
</wsdl:portType>
```

**Figure 3-5 Example of input messages with its descriptive text**

Following the steps of our approach, we need first to extract any useful information from the associated text with each input message in order to find keywords.

The descriptive text for the first message is *FahrenheitToCelsiusSoapIn*, while the descriptive for the second input message is *CelsiusToFahrenheitSoapIn*. We can notice that neither the first input message nor the second can give us any useful keyword that can be used to generate valid and invalid test data, so our approach fail with this example to find any keyword and as a consequence to this failure, it is not possible to generate test data.

However, to handle this situation and any other similar situations, our approach will use the input parameter data types together with constraining facets to generate test data.

It worth mentioning here, generating test data will be based on regular expressions or pre-defined value space and the input parameter data types, in other words test data based on regular expression or pre-defined value space will be generated in parallel with generating test data based on input parameter data types and their associated constraining facets whether the first method succeeded or not in order to reach more test data coverage.

To generate test data based on input parameters and their associated constraining facets, our approach uses test case generation rules.

In order to generate valid and invalid test data our approach uses each data type with its specific constraining facets, the following rules illustrate how test data can be generated for the build-in data type and their associated constraining factes.

If DT = Type1 and CF = MinInclusive or MinExclusive then TD= {Min-1,Min,Min+1}

If DT = Type1 and CF = MaxIclusive or MaxExclusive then TD={Max-1,Max,Max+1}

If (DT = Type1) or (DT = Type2) or (DT = Type3) and CF = Enumeration then TD= {value outside the set,value in the set,value has different data type,null}

If (DT = Type1) or (DT = Type2) or (DT = Type3) and CF = WhiteSpace then TD= {null, tabs, space, multiple spaces}

If (DT = Type1) or (DT = Type2) and CF = pattern then TD= {the value deduced from the pattern, apply the same perturbation strategy in section 5.4 }

If DT = Type1 and CF = Totaldigits or Fractiondigit then TD= {a value exceeds the max allowed number of digits, a value exactly the max allowed number of digits, a value below the max allowed number of digits }

If (DT = Type2) or (DT = Type3) and CF = Length then TD= {string with length-1,string with the exact length, string with length+1}

If (DT = Type2) or (DT = Type3) and CF = MinLength then TD= {string with MinL-1,string with the exact MinL, string with MinL+1}

If (DT = Type2) or (DT = Type3) and CF = MaxLength then TD= {string with MaxL-1,string with the exact MaxL, string with MaxL +1}

Where DT: input parameter data type

CF: constraining facet

TD: test data

Max: the maximum allowed value.

Min: the minimal allowed value.

Type1= Byte || Decimal || Int || Integer || Long || negativeInteger || nonNegativeInteger || nonPositiveInteger || Short || unsignedLong || unsignedInt || unsignedShort || unsignedByte || Double || Float || date || DateTime || Duration || gDay || gMonth || gMonthDay || gYear || gYearMonth || Time

Type2= String || Entity || ID || IDREF || Language || Name || NCName || NMTOKEN || NormalizedString || Notation || Token || AnyURI || QName || Base64Binary || HexBinary

Type3= ENTITIES || IDREFS || NMTOKENS

The generated test data will be saved in XML file, valid and invalid test data will be sent in a SOAP messages to the Web Service under test to assess the its robustness quality attribute. The Web Service under test must respond with a response message for valid inputs or with a fault message with proper fault error string for invalid inputs, otherwise, a robustness problem is encountered.

## 3.9 The basic algorithm

Figure 5.6. Shows the basic algorithm for our approach in pseudo code:

```
Read WSDL file
For each input message in WSDL file {
   Read text associated with input message
   If (found Keyword) {
     If (found Reg. Exp.) generate test data based on Reg. Exp. perturbation rules
```

```
        Else generate test data based on pre-defined value range
      }
}
For each input parameter in WSDL file || XSD file {
    If (found constraining facet) Generate test data based on pre-defined generation rules
}
```

**Figure 3-6 The algorithm**

## 3.10 Summary

This chapter describes the proposed approach for generating test data. This approach is based on the semantics that can be extracted from the descriptive text associated with input messages and on analyzing input parameters data types. Modification strategy was introduce to modify regular expressions in order to generate invalid test data and test data generation rules was discussed for each built-in data type. Finally our approach algorithm was developed in pseudo code.

# CHAPTER FOUR: IMPLEMENTATION ISSUES, EVALUATION AND APPLICATION AREAS

## 4.1 Introduction

In this chapter, we will show the implementation issues for our work. We will also present where to use our approach. Then an evaluation for the correctness and effectiveness of our approach is discussed by applying our approach to a real Web Service.

## 4.2 Implementation issues

The implementation environment of our approach requires any powerful programming language such as JAVA to implement the GUI of our approach, and a compatible data base such as oracle or MySQL to store the test data generated using the methods discussed in the previous chapter.

## 4.3 Application areas:

Web Services testing techniques will be strengthened by adding our approach, since it’s generate more realistic test data than the existing Web Services testing techniques.

Our approach for generating test data can be used in any Web Services domains.

## 4.4 Evaluation

We apply  our algorithm on a real Web Service to evaluate its usefulness in generating test data. The MailBoxValidator Email Validation Web Services. This Web Service provides the ability to check e-mail addresses. This service can be used to reduce deceptive email address in online accounts registration by checking email addresses validity.

Figure (6-1) represents the WSDL file for MailBoxValidator Email Validation Web Services:

```
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tm="http://microsoft
.com/wsdl/mime/textMatching/"xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmln
s:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://ws.fraudlabs.com/"xmlns:s="h
ttp://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" x
mlns:http="http://schemas.xmlsoap.org/wsdl/http/"xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://ws.fraudlabs.com/">
<wsdl:types>
<s:schema elementFormDefault="qualified" targetNamespace="http://ws.fraudlabs.com/">
</xs:schema>
 <xs:element name=" EmailValidator " type="s0: EmailValidator " />
    <xs:simpleType name=" EmailValidator ">
   <xs:restriction base="xs: string ">
    < xs:minLength value="6"/>
    < xs:maxLength value="254"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
<s:element name="EmailValidatorResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="EmailValidatorResult" type="tns:EMAIL_VA
LIDATOR"/>
</s:sequence>
</s:complexType>
</s:element>
<s:complexType name="EMAIL_VALIDATOR">
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="IS_SYNTAX" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="IS_DOMAIN" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="IS_SMTP" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="IS_LEVEL" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="CREDITSAVAILABLE" type="s:string"/>
<s:element minOccurs="0" maxOccurs="1" name="MESSAGE" type="s:string"/>
</s:sequence>
</s:complexType>
<s:element name="EMAIL_VALIDATOR" nillable="true" type="tns:EMAIL_VALIDATOR"/>
</s:schema>
</wsdl:types>
<wsdl:message name="EmailValidatorSoapIn">
<wsdl:part name="parameters" element="tns:EmailValidator"/>
</wsdl:message>
<wsdl:message name="EmailValidatorSoapOut">
<wsdl:part name="parameters" element="tns:EmailValidatorResponse"/>
</wsdl:message>
<wsdl:message name="EmailValidatorHttpGetIn">
<wsdl:part name="EMAIL" type="s: EmailValidator "/>
</wsdl:message>
<wsdl:message name="EmailValidatorHttpGetOut">
<wsdl:part name="Body" element="tns:EMAIL_VALIDATOR"/>
</wsdl:message>
<wsdl:message name="EmailValidatorHttpPostIn">
```

```
<wsdl:part name="EMAIL" type="s: EmailValidator "/>
</wsdl:message>
<wsdl:message name="EmailValidatorHttpPostOut">
<wsdl:part name="Body" element="tns:EMAIL_VALIDATOR"/>
</wsdl:message>
<wsdl:portType name="EmailvalidatorSoap">
<wsdl:operation name="EmailValidator">
<wsdl:input message="tns:EmailValidatorSoapIn"/>
<wsdl:output message="tns:EmailValidatorSoapOut"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="EmailvalidatorHttpGet">
<wsdl:operation name="EmailValidatorOp">
<wsdl:input message="tns:EmailValidatorHttpGetIn"/>
<wsdl:output message="tns:EmailValidatorHttpGetOut"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:portType name="EmailvalidatorHttpPost">
<wsdl:operation name="EmailValidatorOp">
<wsdl:input message="tns:EmailValidatorHttpPostIn"/>
<wsdl:output message="tns:EmailValidatorHttpPostOut"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="EmailvalidatorSoap" type="tns:EmailvalidatorSoap">
.
.
.

</wsdl:binding>
<wsdl:service name="Emailvalidator">
.
.
.
</wsdl:definitions>
```

**Figure 4-1 MailBox Validator WSDL file**

The PORTTYPE element of the MailBox Validator WSDL file consists of three input message, and each input message has one input parameter (EMAIL) of type Emailvalidator, which is of base type “String” and has two constraining facets that restricts its acceptable values (maxlenght and minlenght). When we apply our algorithm on the previous MailBox Validator WSDL file we can have the following test data:

For the three extracted input messages we get the Keyword “Email” from the descriptive text for each message. Then we map the Keyword “Email” to a predefined regular expression “[w- \ .]+@ ([\w-] + \.) + [\w-] {2,4}”, and then use this regular expression to generate valid and invalid test data.

For the input parameter (EMAIL) of type Emailvalidator, we generate valid and invalid test data using the following generation rules:

If (DT = String) CF = MinLenght then TD= {string with MinL-1,string with the exact MinL, string with MinL+1}

If (DT = String) CF = MaxLenght then TD= {string with MaxL-1,string with the exact MaxL, string with MaxL +1}.

Table 6-2 shows the generated test data for the MailBox WSDL file using our approach.

**Table 4-1 Test data for MailBox Validator WSDL file**

| | | |
|---|---|---|
| **Test data generated using Regular Expression** | TestEmail@testcomp.com | |
| | @TestEmailtestcomp.com | |
| | TestEmail@.com | |
| | TestEmailtestcomp.com | |
| | TestEmailtestcomp.com | |
| | TestEmail @testcomp.com | |
| | 2011-0812-072531 | |
| **Test data generated using test data generation rules** | i@g.c | |
| | i@g.cm | |
| | i@g.com | |
| | i…m.com | Email length = 253 |
| | i…m.com | Email length = 254 |
| | i…m.com | Email length = 255 |

## 4.5 Assessing the robustness quality attribute of Web services

In order to assess the robustness quality attribute of Web services, the test data generated in table 6.1 are used.

Each test data is sent to the Web service under test in a SOAP message to evaluate the robustness quality of the Web service, then the response of the Web service is analyzed to determine its behavior in the presence of invalid test data.

The response of the Web service for invalid test data can be one of the following scenarios:

- The Web service rejects the invalid test data and sends a proper error message: for example if we send “2011-0812-072531” as an input value for the input parameter “EMAIL” of the mailbox validator Web service, we assume that the Web service rejects it and sends an error message for example “invalid input data”, for this scenario and every similar scenarios where the Web service under test rejects the invalid inputs, we assume that this Web service is robust.
- The Web service accepts the invalid test data and returns an output: in this scenario a robustness problem is considered since the Web service under test doesn’t behave the way we expected (rejects the invalid test data and return an error message).
- The Web service hangs or crashes when receiving invalid test data: this scenario also arise a robustness problem for the Web service under test since it doesn’t handle invalid test data and moreover it hanged or crashed.

## 4.6 Comparison with similar works

The main difference between our work and other previous similar works is the use of semantics in generating test data. Based on semantics we can compare our work with others through the following comparison criteria’s:

1. Generating realistic test data that are more close to real world problem:
   This thesis has introduced an approach to generate test data based on the semantics that can be extracted from WSDL description of Web Services. The resulting test data are more close to represent what is actually expected to be as an input for a specific input parameter. In the other hand, none of the previous works have been considered this criteria when they generate test data.

2. increasing the test coverage in an efficient and accurate way:

   Previous works such as (De Melo & Silveira, 2011) introduced an approach to generate test data that increases the test coverage but they did not considered test data accuracy. In the other hand our approach was able to generate test data that achieve coverage (by depending on both the semantics and syntax based on the information extracted from WSDL’s specifications only), this added the efficiency to our approach since the generated test data were designed automatically and covers a widely range of test data possibilities. In addition our approach was also able to generate more accurate test data by taking the semantics into account.

3. Generating test data for any Web Service regardless of the programming languages used to build it:

   In contrast of the previous work where they mainly depend on input parameters specifications (input parameters specifications can be different even for the same web service when using different Web services development tools to generate WSDL files) to generate test data, our approach was able to generate test data to assess the robustness quality attribute for **any WSDL**, no matter what programming languages used to generate it by depending on the semantics of the keywords extracted from the input messages inside WSDL.

# CHAPTER FIVE: CONCLUSION AND FUTURE WORKS

## 5.1 Conclusion: perspective and future works

Through our study about Web Services testing, we found that current methods did not use semantics in the process of generating test data. We also found that the generated test data using the current approaches still don't provide a realistic test data that represent the expected input data for the Web Service under test.

In section 1.5, we proposed four contributions to be done during this thesis. the first contribution was achieved by introducing an approach in chapter 5 that can be used to assess the robustness quality attribute of Web Services also we specify how test data can be generated. The second and fourth contribution was done by introducing a new method in generating test data depending on the semantics, using the semantics to generate test data we can now generate test data for any WSDL regardless of the used programming language. And we also successfully generate test data that is better and realistic.

Our work can be extended and developed in future to:

- Assessing other quality attribute of Web Services such as security and performance.
- Finding a way to let our approach learn to choose the appropriate test data generation rule (introduced in section 5.6) without the need to examine each rule for every input parameter data type.
- Extending our approach in a way that can handle complex data types and WSDL's that don't have keywords or constraining facets.
- Finding an automatic way to reduce the effort and time taken in fining keywords.
- Generalizing our approach so we can depend only on WSDL semantics to generate test data.
- Implementing a tool that demonstrates the effectiveness of the proposed Web Services robustness testing approach.

# REFERENCES

Albreshne, A. F. (2009). Web Services Technologies: State of the Art, University of Fribourg, Switzerland. Technical Report No. 09-04, University of Fribourg, Department of Informatics, Switzerland.

ANUPRIYA, J., SHARMA, S., SEEMA, S., & DEEPTI, J. (2010). Boundary value analysis for non-numerical variables: Strings. Oriental Journal of Computer Science & Technology , Vol. 3 (2), 323-330.

Bartolini, C., Bertolino, A., Marchetti, E., & Polini, A. (2009). WS-TAXI: A WSDL-based Testing Tool for Web Services. International Conference on Software Testing, Verification, and Validation (pp. 326–335). Los Alamitos, CA, USA: IEEE Computer Society.

Bashir, R., Azam, F., Aqeel Iqbal, M., Khanum, A., & Malik, H. (2012). A Comparative Model for Tradeoff Analysis of QoS Attributes in SOA. Journal of Basic and Applied Scientific Research .

Bozkurt, M., Harman, M., & Hassoun, Y. (2010). Testing web services: a survey. Department of Computer Science. King’s College London.

Casado, R., Tuya, J., & Younas, M. (2012). Testing the reliability of web services transactions in cooperative application. Proceedings of the 27th ACM Symposium on Applied Computing (SAC 2012). Riva, Trento, Italy.

Cavanaugh, E. (2006). Web services: Benefits, Challenges, and a Unique Visual Development Solution. USA: Altova White Paper.

De Melo, A. C., & Silveira, P. (2011). Improving data perturbation testing techniques for Web services. Information Sciences , 181 (3), 600-619.

De Virgilio, R. (2010). Meta-Modeling of Semantic Web Services. Services Computing (SCC),2010 IEEE International Conference on, (pp. 162-169).

Dragoni, N. (2009). Toward Trustworthy Web Services - Approaches, Weaknesses and Trust-By-Contract Framework. Proceeding WI-IAT '09 Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology. 03, pp. 599-606. Washington, DC, USA : IEEE Computer Society .

Fu, J., Hao, W., Bastani, F. B., & Yen, I.-L. Y. (2011). Model-Driven Development: Where Does the Code Come From? , Insights Learned From a Case Study. Fifth IEEE International Conference on Semantic Computing.

Hanna, S. (2008). Web Services robustness testing, Ph.D, Durham theses, Durham University.

Hanna, S., & Abu Ali, A. (2011). Platform Effect on Web Services Robustness Testing. Communications of the Applied Computer Science Journal , 11 (2), 360- 366.

Hanna, S., & Alawneh, A. (2010). An Approach of Web Service Quality Attributes Specification. 2010. Communications of the IBIMA.

Hanna, S., & Munro, M. (2009). An Approach for WSDL-Based Automated Robustness Testing of Web Services. Information Systems Development: Challenges in Practice, Theory, and Education. vol. 2, pp. 493-504. Springer.

Huang, W.-l., & Peleska, J. (2013). Exhaustive Model-Based Equivalence Class Testing. The 25th IFIP International Conference on Testing Software and Systems, ICTSS2013, Lecture Notes in Computer Science. vol(8254), pp. 49-64. Springer.

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society.

Jorgensen, P. C. (2013). Software Testing: A Craftsman's Approach, Fourth Edition. CRC Press.

Khan, M. E. (2012). A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. Vol. 3, No.6.

Khan, M. E. (2011). Different Approaches to White Box Testing Technique for Finding Errors. International Journal of Software Engineering and its Applications , vol. 5, no. 3.

Kumar, S., & Varalakshmi, P. (2012). Dynamic Web Service Composition based on Network Modeling with Statistical Analysis and Backtracking. International Journal on Web ervice Computing (IJWSC) , 3 (2).

Laranjeiro, N., Vieira, M., & Madeira, H. ( 2014). A Technique for Deploying Robust Web Services. Services Computing, IEEE Transactions on. vol.7, no.1, pp. 68-81. IEEE.

Lee, C.-H., & Hwang, S.-Y. (2009). A Model for Web Services Data in Support of Web Service Composition and Optimization. Services - I, 2009 World Conference on (pp. 384 - 391). Los Angeles, CA: IEEE.

Li, N., Xie, T., Jin, M., & Liu., C. (2010). Perturbation-based user-input-validation testing of web applications. Journal of Systems and Software (JSS) , 83(11), 2263–2274.

Makhlughian, M., Hashemi, S. M., Rastegari, Y., & Pejman, E. ( 2012. ). WEB SERVICE SELECTION BASED ON RANKING OF QOS USING ASSOCIATIVE CLASSIFICATION. the International Journal of Web Service Computing (IJWSC) , 3 (1).

Moreira, A. M., Antunes, C. H., & Ramal, d. M. (2010). Application of a Syntax-based Testing Method and Tool to Software Product Lines. National Institute of Science and Technology for Software Engineering-INES .

Noikajana, S., & Suwannasart, T. (2009). An improved test case generation method for web service testing from wsdl-s and ocl with pair-wise testing technique. Computer Software and Applications Conference, COMPSAC '09. 33rd Annual IEEE International. volume 1, pp. 115 - 123. IEEE.

O'Brien, L., Merson, P., & & Bass, L. (2005). Quality Attributes and Service-Oriented Architectures. (CMU/SEI-2005-TN-014), Carnegie Mellon University, Software Engineering, Pittsburgh.

Rani, B. P., Rao, K. V., & Devi, K. M. (2010). Architecting Secure Web Services using Model Driven Agile Modeling. International Journal of Engineering Science and Technology , 2 (9), 4603-4609 .

Rayns, C., Burgess, G., Cooper, P., Fitzgerald, T., Goyal, A., Klein, P., et al. (2010). Application development for CICS Web services (2nd ed.). The Free Library, IBM - Int'l Tech Support Org.

Salva, S., & Rabhi, I. (2009). Automatic Web Service Robustness Testing from WSDL descriptions. In Proceedings of the 12th European Workshop on Dependable Computing (EWDC '09). Toulouse, France.

Shah, M., Verma, Y., & Nandakumar, R. (2012). AN AUTOMATED END-TO-END MULTI-AGENT QOS BASED ARCHITECTURE FOR SELECTION OF GEOSPATIAL WEB SERVICES. XXXIX-B4.

Sun, W., Li, S., Zhang, D., & Yan, Y. (2009). A Model-Driven Reverse Engineering Approach for Semantic Web Services Composition. Software Engineering, 2009. WCSE '09. WRI World Congress on. 3, pp. 101-105. Xiamen: IEEE.

W3C. (2010, May). Web Services description language (WSDL), 2 part 1: Core language. Retrieved May 2010, from <http://www.w3.org/TR/wsdl20/>

W3C. (2010, May). Web Services glossary. Retrieved May 2010

W3C. (2008, December 9). XML schema part 2 : Dtatypes. Retrieved from W3C Recommendation .

# ملخص

تعتبر خدمات الويب من اهم التطبيقات المستخدمة حديثا وتأتي هذه الاهمية من قدرتها على التكامل والاتصال مع التطبيقات الموزعة داخل شبكة الانترنت بعض النظر عن البيئة الغير متجانسة التي تعمل بها هذه التطبيقات وكذلك بعض النظر عن لغات البرمجة المستخدمة لبناء هذه التطبيقات.

وعلى الرغم من الانتشار الواسع لخدمات الويب فأن هناك بعض المشاكل التي تحد من انتشارها واعتمادها من قبل الاشخاص والمنظمات لتكون مكون اساسي في اعمالهم. واحده من هذه المشكلات هي صعوبة وعدم قدرة الطالب للخدمة لتحديد ما اذا كانت الخدمة المقدمة من مقدم الخدمة قابلة للثقه لديه ام لا وبتالي عدم قدرته على تحديد مستوى جودة الخدمة المقدمة .

طريقة مثلى لزيادة مستوى الثقة ما بين مقدم الخدمة ومستخدم الخدمة هي باستخدام الفحص للخدمة قبل طلبها من قبل طالب الخدمة، حيث يعطي الفحص طريقة اتوماتيكية لتقييم نوعية خاصية المتانه لخدمة الويب المقدمة وقياس الطريقة التي تتعامل بها الخدمة في حالة وجود مدخلات خاطئة. عملية الفحص تتطلب فقط وجود ملف وصف الخدمة ( WSDL ) حتى يتمكن طالب الخدمة من قياس مدى نوعية خاصية المتانه لخدمة ويب معينة.

تقدم هذه الاطروحة نهج جديد بحيث يمكن استخدامة من قبل طالب الخدمة لتقييم وقياس نوعية خاصية المتانه للخدمات الويب.

النهج المتبع في هذه الاطروحة مبني علي تحليل وثائق لغة وصف خدمات الويب ( WSDL ) ومن ثم بناء حالات اختبار بناء على المعنى الذ يمكن استخراجه من رسائل الادخال الموجودة داخل ملف لغة وصف خدمات الويب وكذلك على نوع البيانات لمتغيرات الادخال ومن ثم استخدام حالات الاختبار المنشئة لتقييم نوعية خاصية المتانه لخدمة الويب.

هذا النهج الجديد سيعطي حالات اختبار افضل تمثل الخدمة المقدمة من خدمات الويب بشكل اكثر واقعية واكثر منطقية كما انه يعمل على فحص نوعية الخصائص لخدمات الويب بشكل اكثر فاعلية.

# تقييم نوعية خاصية المتانه لخدمات الويب بالاعتماد على الدلالات المستخرجه من لغة وصف خدمات الويب

بواسـطة

مروان فايز عبدالله المومني

بإشـراف

د. سامر حنا

قدمت هذه الرسالة استكمالاً لـمـتطلبات الحصول على درجة

الـمـاجستير في علـم الـحـاسوب

عمـادة البحث العلمي والدراسات العليا

جامعة فيلادلفيا

آب 2014